BIBLIOTECA BASICA INFORMATICA

FORTRAN Y COBOL

31

a su disposición



INGELEK

BIBLIOTECA BASICA INFORMATICA

FORTRAN Y COBOL

31

a su disposición

INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Monsalve.

Redactor técnico:
Alfredo B. García Pérez.

Colaboradores:
Casimiro Zaragoza.

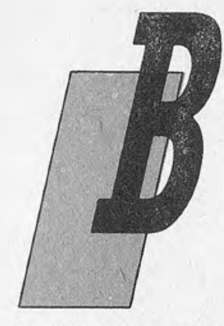
Diseño:
Bravo/Lofish.

Dibujos:
José Ochoa.

ISBN del tomo: 84-85831-64-0
ISBN de la obra: 84-85831-31-4
Fotocomposición Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito legal: M-1990-1986
Precio en Canarias, Ceuta y Melilla: 380 pts.

INDICE

PROLOGO



Bastantes de los programas existentes hoy en día nacieron con el propósito de mantener una flexibilidad y adaptabilidad que les permitieran tener aplicación en la mayor parte de los campos posibles. El ejemplo más representativo al respecto es el BASIC. Sin embargo, aunque han tenido y tienen una gran difusión, no han logrado ese objetivo; se han quedado tan sólo en el «servir un poco para todo pero mucho para nada», lo que les excluye de trabajos donde se requiere una gran potencia en determinadas áreas de la programación.

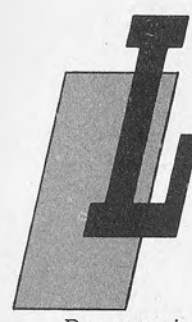
Este hueco es cubierto precisamente por unos lenguajes mucho menos difundidos y conocidos, que tienen un campo muy limitado donde intervenir, pero que en él desarrollan toda su capacidad y potencia. Claras muestras de ellos son el FORTRAN (FORmula TRANslation) y el COBOL (COmmon Business-Oriented Language).

El primero destaca especialmente en el campo de la computación científica. Su u primera versión (FORTRAN I) nació en 1956. Actualmente se usa mayoritariamente el FORTRAN-77.

El COBOL, por contra, se centra en las actividades comerciales, siendo capaz de un manejo eficaz de datos. Utilizado por primera vez en los años sesenta, la versión más extendida es la 66, aunque el COBOL-80 está "a punto de caramelo".

CAPITULO I

LOS LENGUAJES



a transmisión de las informaciones al interior de un ordenador se obtienen mediante la codificación de las señales eléctricas intercambiadas entre los diversos dispositivos que constituyen la estructura funcional.

En realidad, la señal eléctrica elemental puede tomar solamente dos niveles, a los cuales se les asocian los estados "0" y "1", respectivamente.

Por consiguiente, si se trabaja con una sola señal únicamente se podrían intercambiar dos informaciones. Es evidente, pues, que doblando la complejidad de los circuitos se podrán tratar cuatro informaciones diferentes (00, 01, 10, 11), triplicándola se podrán tratar ocho informaciones (000, 001, 010, 011, 100, 101, 110, 111) y así sucesivamente, siguiendo el esquema lógico de que a un número "n" de circuitos corresponderán 2^n informaciones "binarias".

Un ordenador, para poder funcionar, tiene la necesidad de reconocer informaciones de "tipo" diverso, por cuanto que ha de poder distinguir si una combinación de cifras binarias corresponde a un comando a ejecutar, al dato que ha de procesar o a la dirección (la posición "geográfica") de una celda de memoria.

Esto es así porque un ordenador, además de ejecutar los comandos que se le dan uno tras otro, está en condiciones de "digerir" una lista completa y de controlar, de forma autónoma, el modo en que seguir la lista de las instrucciones o, lo que es lo mismo, el "programa" que se le proporciona.

En efecto, el diálogo con los primeros ordenadores se realizaba mediante la introducción "material" de secuencias de "0" y "1" en las celdas destinadas a contener las instrucciones de pro-

grama, lo que limitaba enormemente la dimensión de los procedimientos realizables.

Este código de diálogo directo con el ordenador se denomina código máquina.

Evolución

Más adelante, las reglas del "lenguaje" experimentaron algunos cambios para permitir una mayor agilidad en la programación; se asignaron nombres simbólicos a las posiciones de memoria y se condensaron en algunos símbolos las características de cada operación. Por ejemplo, la instrucción:

ADD A, 4

tenía el significado de añadir el valor 4 al contenido de la celda de memoria definida como A y de conservar el resultado de la suma en dicha celda. Un programa escrito según una sintaxis semejante es mucho más fácil de manejar por cuanto que presenta relaciones mnemónicas entre la acción que se quiere realizar y el conjunto de caracteres que la representa, pero tiene necesidad de traducirse a código máquina para que lo comprenda el ordenador. Esta traducción se realiza, a su vez, por un programa denominado ensamblador, el cual lee una a una las líneas del programa fuente y las transforma en el correspondiente conjunto de instrucciones en binario, creando el denominado código objeto.

Este último será, en una fase sucesiva, el programa efectivamente ejecutado. La instrucción "ADD A, 4" se transformaría una vez traducida, por ejemplo, en el conjunto de dígitos binarios 11000110 00000100.

No obstante, el lenguaje ensamblador es característico de cada máquina particular y ello hacía bastante difícil la definición de códigos mnemónicos universalmente utilizables y dificultaba el empleo del mismo programa en ordenadores diferentes.

Una posterior evolución de los lenguajes de programación ha puesto de manifiesto la necesidad de dialogar con los ordenadores en formas más próximas a los modos de expresión de los seres humanos. Así nacieron los lenguajes de alto nivel, en los cuales las instrucciones siguen reglas sintácticas similares a las de las lenguas habladas (sobre todo del inglés) y, para las operaciones de tipo matemático, las reglas algebraicas, mientras que los comandos se expresan mediante verbos y vocablos que indican directamente el significado de la operación a realizar. En un lenguaje de alto nivel el programador no tiene necesidad alguna de

conocer la estructura interna del ordenador en que trabaja, pero puede utilizar el mismo programa en ordenadores diferentes con la certeza de obtener efectos idénticos.

Por ejemplo, la instrucción FORTRAN

A = B + 1

producirá el efecto de sumar 1 al valor contenido en la posición de memoria indicada por B y de conservar el resultado en la posición identificada por A, con la ventaja adicional de reproducir las notaciones del álgebra y, por consiguiente, de ser fácilmente comprensible.

La instrucción COBOL

MOVE ZERO TO VAR_A

hará que la celda de memoria correspondiente a la variable VAR_A contenga el valor 0. Ello sucederá así en cualquier ordenador en el que se apliquen programas que contengan dichas instrucciones, por cuanto que los efectos producidos dependen, casi en su totalidad, de la organización sintáctica de los lenguajes y no de las estructuras funcionales de los ordenadores.

La tarea de la traducción de instrucciones de alto nivel a código objeto es especialmente onerosa y los programas que la realizan, muy sofisticados y evolucionados, se conocen como intérpretes o compiladores, dependiendo del mecanismo de transformación que utilicen.

Los compiladores trabajan como los ensambladores, por cuanto que la fase de traducción de las instrucciones del programa fuente a código objeto y la de la ejecución efectiva son distintas. En la fase de traducción, además, se realizan también verificaciones de la corrección sintáctica de las instrucciones. La compilación exige mucho tiempo, pero el programa objeto resultante es muy rápido.

En los intérpretes, por el contrario, las dos fases son simultáneas, por cuanto que controlan directamente la ejecución del programa. En efecto, las instrucciones, una tras otras, son objeto de lectura, se decodifican según las indicaciones sintácticas, se traducen a las instrucciones elementales correspondientes y, finalmente, se ejecutan. Si bien trae consigo la repetición de ciclos de operaciones (ya que el mismo comando se decodifica cada vez que se le encuentra), permite una puesta a punto del programa más fácil, habida cuenta de que la corrección de un error se efectúa de manera inmediata.

Tipos de lenguajes

Desde los años cincuenta hasta el momento presente han nacido muchos lenguajes; algunos de ellos con el objeto de permitir que el programador sea completamente independiente de las estructuras de los ordenadores, otros orientados a la resolución de los problemas que plantean necesidades particulares de los usuarios, otros concebidos para mejorar el estilo de programación y la estructuración lógica del análisis de un problema y otros (los más jóvenes) tienen la pretensión de constituir "el lenguaje universal".

Entre los del primer tipo recordamos a los lenguajes ALGOL y FORTRAN, nacidos con marcadas connotaciones de tipo científico; el COBOL y el RPG, muy apreciados en el ámbito comercial; el PL/1, que constituye una tentativa de conciliación de las exigencias de los dos campos anteriores, y el BASIC, bastante más reciente, concebido para fines didácticos, pero acogido con tal entusiasmo en el ámbito de los ordenadores personales que ha llegado a alcanzar un grado de difusión que se podría considerar como universal.

En el segundo grupo se pueden citar: el APT, utilizado para la descripción de los movimientos de las herramientas en las máquinas de control numérico; el STRESS, para la resolución de problemas de ingeniería estructural; el SIMULA; para los modelos de simulación; el GTL; para problemas de tipo geométrico y tecnológico y los lenguajes adaptados a la preparación de sesiones de aprendizaje, tales como TUTOR, PILOT y COURSE-WRITER.

En el tercer grupo están incluidos los lenguajes de tipo estructurado, tales como PASCAL, FORTH, PROLOG y LISP, algunos de los cuales son idóneos para abordar los problemas de los sistemas expertos y los relativos al mundo de la inteligencia artificial. Estos lenguajes tienen una forma sintáctica muy similar a la utilizada para expresar los resultados de un análisis.

En el último grupo podemos incluir el lenguaje C, potente y complejo, que permite la "portabilidad" casi total entre ordenadores diferentes, y el ADA, que tiene su origen en el Departamento de Defensa de los Estados Unidos, nacidos ambos con el objetivo reivindicado de constituir patrones universales.

FORTRAN

Las bases para el nacimiento del Fortran fueron puestas en el año 1954 por el equipo de trabajo de John Backus (IBM) que, valorando las dificultades planteadas por la programación en ensamblador, decidió obtener un compilador más flexible que fuese uti-

lizable por un ordenador del tipo 704. Nació así el denominado IBM Mathematical FORmula TRANslation system, es especialmente apto en la formulación de procedimientos para la resolución de problemas de tipo científico.

Muy pronto constituyó un punto de referencia para productos análogos de los otros fabricantes de ordenadores y para la propia IBM que, impulsada por los poseedores de modelos diferentes del 704, realizó versiones del compilador Fortran utilizables para los diversos modelos de sus ordenadores. A causa de la existencia de varias decenas de compiladores Fortran, todos ellos incompatibles entre sí, al comienzo de los años sesenta el ANSI (American National Standards Institute) puso en marcha el proceso de unificación de las características del lenguaje para depurarlo de las peculiaridades presentadas por las diferentes máquinas. En el año 1966 apareció de forma oficial la versión definitiva estándar, denominada Fortran 66.

En los años sucesivos, el gran desarrollo de la cultura informática, la difusión de los ordenadores en campos en donde la naturaleza de los problemas favorece el aspecto de gestión sobre el aspecto científico y el nacimiento de lenguajes especializados de mayor potencia hicieron necesarias modificaciones de la versión oficial de Fortran.

Iniciada en 1969 y terminada en 1977, la revisión de las características del lenguaje estándar definió el Fortran 77, versión que con respecto a las anteriores presenta el potenciamiento de algunas funciones y la posibilidad de tratamiento de cadenas de caracteres, lo que antes era prácticamente imposible.

La facilidad de empleo y la capacidad de definir funciones matemáticas complejas constituyen los puntos fuertes del Fortran; treinta años de vida activa han sido testigos de la realización de proyectos complejos en los campos industrial y aeronáutico (Boeing, Apt y Nasa) y de la difusión de instrumentos operativos "escritos" en Fortran en el campo estadístico, gráfico, en la resolución de problemas de análisis numérico y estructural, en la metodología de los modelos y en la investigación en general.

Por otra parte, una de las características principales del Fortran es la de poder escribir *subrutinas*, es decir, pequeños conjuntos de instrucciones aptas para resolver problemas recurrentes, susceptibles de activación por cualquier prograna y que constituyen un banco de software propiamente dicho.

Por otra parte, se han realizado muchas bibliotecas (colecciones de subprogramas) que permiten abordar los problemas típicos de cada área de aplicación; así quien tenga necesidad de escribir un nuevo programa podrá tener acceso a una gran cantidad de subrutinas ampliamente probadas, lo que le supondrá un gran ahorro de tiempo en la escritura y pruebas correspondien-

tes. Habida cuenta de la gran cantidad de programas existentes en Fortran no será fácil suplantarle en el campo científico a no ser por un lenguaje que presente características evidentemente superiores en flexibilidad y potencia.

COBOL

El Cobol es un gigante en el campo de los lenguajes de alto nivel; su difusión a nivel empresarial, el gran número de procedimientos escritos y en funcionamiento y los tiempos y el trabajo destinados a hacerles operativos son datos que será preciso tener en cuenta en la hipótesis de su sustitución por lenguajes todavía mejor estructurados.

El Cobol (Common Business Oriented Language) nació en 1960 para resolver problemas de gestión. La comisión de representantes del Pentágono de Estados Unidos, de fabricantes de ordenadores y de usuarios que patrocinó las primeras versiones dio también vía libre a un proceso de actualizaciones continuas que ofreció como resultado final el COBOL ANSI 74 en el año 1974.

Los problemas de gestión se diferencian de los problemas científicos en el gran número de datos implicados y en la menor importancia de las fases de proceso propiamente dicho. El compilador Cobol está dotado de un potente conjunto de instrucciones para la gestión de datos de diversa naturaleza, tales como variables y constantes, denominaciones simbólicas complejas, reagrupamientos, vectores, tablas, registros y ficheros. En particular, los ficheros, que son conjuntos de datos rígidamente estructurados en registros y campos, pueden procesarse según diversas modalidades, que van desde el acceso secuencial, que permite tratar todos los registros uno tras otro, a la organización con índice, que permite recuperar todas las informaciones contenidas en un registro mediante la indicación de su clave de acceso.

Además, es posible controlar simultáneamente varios puestos de trabajo y tener acceso a los propios ficheros por parte de varios usuarios, así como transmitir e intercambiar datos entre diferentes unidades.

Por otra parte, resulta prácticamente imposible resolver un problema científico con un programa en Cobol por la ausencia de funciones tales como las trigonométricas, matemáticas, etc.

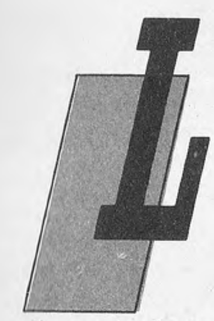
Un sector en el que el Cobol resulta ser de gran utilidad es la gestión de tablas y la impresión de informes, es decir, en la presentación de los datos en forma organizada, que se facilita mucho por las definiciones de los tipos de datos que considera el lenguaje.

Otras características peculiares son la rigidez del lenguaje que, no obstante, favorece la "portabilidad" entre sistemas diferentes y la autodocumentación de los procedimientos, debido a la puntual y sofisticada sintaxis (en lengua inglesa) de todas las instrucciones, que se acerca a la prosa.

En la práctica, el Cobol se utiliza para actividades en las cuales los archivos de datos, normalmente de grandes dimensiones, están sometidos a actualizaciones y manipulaciones monótonas y repetitivas, tal como sucede en los procedimientos de contabilidad, pagos e ingresos, gestión de almacenes, de bibliotecas y de hospitales y para las operaciones de introducción de datos de gran magnitud.

CAPITULO II

COMPILADORES



Los programas escritos en lenguaje de alto nivel, frente a su innegable ventaja de ser fácilmente manejables, legibles, controlables y autodocumentados, presentan la característica de producir, una vez compilados, un código objeto ligeramente más "engorroso" que el programa correspondiente escrito en código máquina. Ello es explicable considerando que a cada comando del lenguaje de alto nivel corresponde en código máquina una subrutina que debe conservar la característica de generalizarse y tener en cuenta todas las posibles excepciones, por lo que en la mayor parte de los casos resuelve el problema para el que se creó con un gran número de instrucciones y, por consiguiente, exige un tiempo mayor de ejecución.

En realidad, el precio a pagar, en términos de tiempo de proceso, haría necesaria la valoración, caso por caso, del lenguaje preferible para cada parte del programa. Por otro lado, un programa escrito en muchos de los lenguajes actuales está en condiciones de "llamar" a un código objeto producido por la compilación de programas fuente escritos en otros lenguajes; esta posibilidad se va extendiendo cada vez más a los lenguajes de las nuevas generaciones.

El problema común de los compiladores

La lógica de programación y la estructura de un ordenador imponen que cualquier lenguaje, instrucción por instrucción, proporcione siempre la respuesta a cuatro interrogantes:

- de qué naturaleza es el dato a procesar;
- dónde se encuentra;
- a qué operación debe someterse;
- en dónde conservar el resultado de la operación.

Por consiguiente, en cada comando se deberán encontrar las indicaciones relativas a cada una de ellas.

La memoria de un ordenador es un gigantesco casillero y cada una de las celdas que lo constituyen está identificada por su dirección en el interior del mapa que describe su estructura.

A algunas celdas se les atribuye un significado particular porque en ellas están depositadas informaciones de tipo general necesarias para el funcionamiento de todo el sistema y se le prohibe al usuario la posibilidad de modificar su contenido.

A otras celdas se les atribuye el cometido de "aparcamiento" de informaciones proporcionadas por el usuario, pero el destino de su contenido está preestablecido por el ordenador.

Por último, otras celdas pueden ser controladas directamente por el usuario.

Un programa rellena, pues, en el interior de la memoria de un ordenador, un conjunto particular de celdas cuyo contenido se lee, examina e interpreta según los principios de funcionamiento del ordenador y del lenguaje.

Por el contrario, en lo que respecta al tipo de dato a procesar, es necesario recordar que todas las celdas de memoria tienen la misma capacidad y, por ello, contienen información de la misma longitud, aunque los datos, dependiendo de su propia naturaleza, pueden ocupar una o varias posiciones de memoria.

El esquema de representación "binaria" más utilizado en los ordenadores es el que corresponde a 8 unidades de información (cada unidad, 1 bit, puede valer 0 ó 1) con 256 (2^8) combinaciones diferentes, definiéndose a un grupo de 8 bits con el nombre de byte.

La mayor parte de los ordenadores, al ser 256 un número suficiente para la representación de los símbolos más frecuentemente utilizados (alfabéticos, numéricos, caracteres especiales de puntuación, etc.), adopta la dimensión mínima de celda de memoria igual a 1 byte.

En condiciones normales en un programa, las informaciones están en la forma de cadenas de caracteres (por ejemplo, la frase "ésta es una cadena") o de números de diferente tipo (números enteros, de coma flotante, de doble precisión) y cada una de ellas ocupa una cantidad distinta de celdas de memoria.

Puesto que las informaciones son controladas mediante variables simbólicas de las cuales, en definitiva, se procesa el contenido, se hace indispensable conocer la longitud del espacio ocupado por cada dato, por cuanto que el programa podrá llevar así todos los datos (y las variables) a una zona especial en celdas de memoria contiguas y referirse a cada uno de ellos especificando simplemente la dirección en donde puede encontrarse la primera celda que les corresponde. El número de celdas sucesivas que le pertenecen está en correspondencia directa con el tipo de dato.

Examinemos el programa:

```
A = 2
B = 5
C = A + B
```

con el cual efectuamos una suma entre los valores atribuidos a dos variables y conservamos el resultado en una tercera.

Dentro de la zona de memoria destinada a los programas, los comandos se transformarán en las instrucciones siguientes:

- Primer comando

Celda nº 1 - Inserta el valor 2 en la celda cuya dirección está especificada en las celdas siguientes (2 y 3) y que se refiere a una variable entera.
Celda nº 2 - Primera parte de la dirección de la variable.
Celda nº 3 - Segunda parte de la dirección de la variable.

- Segundo comando

Celda nº 4 - Inserta el valor 5 en la celda cuya dirección está especificada en las celdas siguientes (5 y 6) y que se refiere a una variable entera.
Celda nº 5 - Primera parte de la dirección de la variable.
Celda nº 6 - Segunda parte de la dirección de la variable.

- Tercer comando

Celda nº 7 - Lee el valor contenido en la celda cuya dirección está especificada en las celdas siguientes (8 y 9).
Celda nº 8 - Primera parte de la dirección.
Celda nº 9 - Segunda parte de la dirección.
Celda nº 10 - Suma el valor contenido en la celda cuya dirección está especificada en las celdas siguientes (11 y 12).
Celda nº 11 - Primera parte de la dirección.
Celda nº 12 - Segunda parte de la dirección.

Celda nº 13 - Inserta el valor resultante en la celda cuya dirección está especificada en las celdas siguientes (14 y 15) y que se refiere a una variable entera.
Celda nº 14 - Primera parte de la dirección.
Celda nº 15 - Segunda parte de la dirección.

Si el tipo de variables hubiera sido diferente, por ejemplo de coma flotante, cada una de ellas hubiera ocupado cuatro celdas; el contenido de las celdas 5, 6 y 11, 12 hubiera estado en la dirección 20 y el de las celdas 14 y 15 estaría en la dirección 24.



Figura 1.— Indicación y localización de las variables de un programa en la memoria.

Una última observación aclarará el motivo de la diferente ocupación de memoria de los distintos tipos de datos.

Una cadena de caracteres al estar representado cada carácter por un byte, ocupará tantas celdas como caracteres tenga (incluyendo los espacios), precedidos por un byte que contiene la longitud de la cadena y que indica al compilador el número de celdas ocupadas.

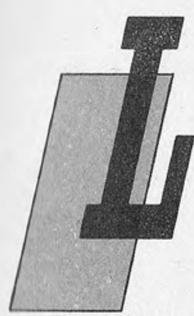
Los números enteros se suelen conservar en dos bytes, lo que permitiría tratar los números desde 00000000 00000000 a 11111111 11111111, es decir, desde 0 a 65565. Para aprovechar mejor la circuitería del ordenador se prefiere considerar negativos todos los números que tienen el primer bit igual a 1, pero leyéndolos como si fueran las cifras de un cuentakilómetros girado al contrario, por lo que 11111111 11111111 corresponderá a -1, 11111111 11111110 corresponderá a -2, y así sucesivamente hasta -32768 (10000000 00000000). Los números positivos tendrán como límite superior +32767.

Para los números reales, en los cuales la presencia de la coma hace problemático el tratamiento de las cifras significativas, se conserva el mecanismo de considerar negativos los números con el primer bit igual a 1, se desplaza la coma hasta obtener un número comprendido entre 0 y 1 y se utilizan los primeros bytes para conservar las cifras significativas y el último para conservar el número de cifras en las que se desplazó la coma. Por consiguiente, el número 124,546 se conservará como 0,124546 x 10^3. La cantidad 124546 se conservará en los primeros bytes y 03 en el último. Según la precisión que se quiera obtener y, por consiguiente, el número de cifras significativas que se quieran conservar, se utilizarán más o menos bytes. Los números conservados con "doble precisión" suelen utilizar 8 bytes.

CAPITULO III

FORTRAN

Variables



Las variables utilizadas en Fortran se identifican por un nombre alfabético o alfanumérico cuya longitud puede extenderse normalmente de 1 a 6 caracteres.

Pueden ser de diversos tipos:

Enteras	INTEGER
Reales	REAL
Doble precisión	DOUBLE PRECISION
Complejas	COMPLEX
De caracteres	CHARACTER
Lógicas	LOGICAL

Solamente las dos primeras son usadas por defecto. A falta de indicación, el Fortran reconoce como enteras todas las variables especificadas por nombres que comiencen por I, J, K, L, M y N, y como reales, todas las que comiencen por otra letra del alfabeto (un nombre de variable comienza siempre con una letra).

Este modo de reconocer las variables puede modificarse o integrarse. Si se quisiera especificar, por ejemplo, como enteras, contrariamente a sus definiciones implícitas, las variables que comienzan con A, B, C y F, se tendrá que utilizar:

```
IMPLICIT INTEGER (A-C,F)
```

mientras que para introducir los demás tipos de variables o para definir una variable como perteneciente a una categoría determi-

nada se tendrán que utilizar las definiciones deseadas en correspondencia con cada una de ellas. Por consiguiente:

```
REAL JOTA
DOUBLE PRECISION HACHE
COMPLEX CALC1, CALC2
CHARACTER*10 CADENA
```

comunicará al compilador que la variable JOTA es real, HACHE es de doble precisión, CALC1 y CALC2 son complejas y CADENA es una cadena alfanumérica de 10 caracteres.

Los valores de doble precisión difieren de los reales por el hecho de contener un número mucho mayor de cifras significativas, mientras que los números complejos son valores identificados por sus coordenadas (parte real y parte imaginaria) en un plano. Las variables CHARACTER son conjuntos de caracteres de los cuales es preciso declarar la dimensión si es mayor que 1.

Las variables lógicas pueden tomar solamente los valores:

.TRUE. (Verdadero)
.FALSE. (Falso)

correspondientes a -1 y 0, y se utilizan para valorar la condición de expresiones particulares.

Vectores y Matrices

Cuando se quiere trabajar con listas de valores del mismo tipo, identificables cada uno mediante el número de orden de la posición que ocupa en la lista, se habla de vectores.

La especificación de un vector es necesaria porque el compilador tiene la necesidad de conocer el espacio ocupado por el bloque completo de datos; así:

```
DIMENSION SERIE(20)
```

especifica que se utilizará en el curso del programa un conjunto de variables reales (por defecto) denominadas SERIE(1), SERIE(2), SERIE(20) que, en memoria, identifican un bloque de 80 posiciones consecutivas (20*4). Si deseamos especificar el número y el tipo de los elementos de un vector es suficiente la indicación de su categoría, lo que permite ahorrar una instrucción. Por consiguiente:

```
INTEGER SERIE(15)
```


tendrá el doble objeto de especificar que se dimensiona un vector de 15 elementos y que se trata de variables enteras.

Naturalmente, se puede trabajar con vectores de todos los tipos incluidos en el Fortran, a condición de especificar cada vez la categoría, el nombre y el número de elementos.

No es obligatorio considerar la numeración de los elementos de un vector desde 1 al número máximo, por cuanto que los límites pueden redefinirse. Así:

```
INTEGER ESTADO (-10:10)
DOUBLE PRECISION CUENTA (0:20)
```

indicarán un vector de valores enteros accesibles mediante un índice que va desde -10 a 10, el primero, y un vector de valores de doble precisión con un índice que varía desde 0 a 20, el segundo.

También es posible trabajar con vectores multiíndice (matrices), es decir, con varios niveles, en los cuales un elemento se reconoce por los valores tomados por cada uno de los índices. El caso de índices es lo que se conoce como matriz bidimensional.

Consideremos, por ejemplo, un archivo de 1000 fichas, cada una de las cuales está subdividida en 15 líneas y cada línea dividida en 8 columnas. El contenido de la sexta columna de la cuarta línea de la ficha 201 será, entonces, identificado por el elemento

```
FICHA (201,4,6)
```

El número máximo de dimensiones es 7.

Los elementos de un vector multidimensional están dispuestos en memoria de modo que se colocan como primeros los elementos con el índice que cambie con mayor rapidez y como últimos los que tengan el índice que cambia con menor rapidez. Esto significa que los elementos del vector

```
FICHA (2,3,2)
```

estarán dispuestos así:

```
FICHA(1,1,1), FICHA(2,1,1), FICHA(1,2,1), FICHA(2,2,1),
FICHA(1,3,1), FICHA(2,3,1), FICHA(1,1,2), FICHA(2,1,2),
FICHA(1,2,2), FICHA(2,2,2). FICHA(1,3,2), FICHA(2,3,2).
```

Asignaciones de valores

Cada variable utilizada en un programa representa simbólicamente el valor contenido en las celdas de memoria que identifica. Es evidente, pues, que este valor debe estar bajo control, o

sea: ha de ser posible asignar a la variable el valor deseado en cada momento. La instrucción general que consigue este objetivo tiene la forma:

variable = valor

Por ejemplo, con:

IVAR1 = 4.5
IVAR = 3*IVAR0

se expresa la intención de introducir el valor 4.5 en las celdas marcadas por la variable real VAR1 y el resultado del producto de 3 por el valor de IVAR0 en las celdas representadas por IVAR.

Cuando se tenga que atribuir valores a los elementos de un vector, este método sería bastante incómodo; por ello se prefiere utilizar sistemas de asignación "global".

Por ejemplo, para el vector de variables reales PRUEBA(10), se podría escribir

DATA PRUEBA(1), PRUEBA(2), PRUEBA(3)/4*12.5/

para indicar que los tres primeros elementos del vector toman un valor igual a la expresión encerrada entre las barras.

Todavía mejor es la expresión:

DATA(PRUEBA(I),I=1,3)/4*12.5/

para indicar lo mismo; significa que el compilador deberá considerar, para cada valor de I desde 1 a 3 inclusive, las variables PRUEBA(I) correspondientes y asignar a estas últimas los valores resultantes de la expresión indicada.

Si se quiere atribuir al mismo tiempo que esto el valor 0 a la variable entera J, y la palabra PRUEBA a la variable

CHARACTER * 6 LETR,

se tendrá que escribir:

DATA (PRUEBA(I),I=1,3)/4*12.5/J/0/,LETR/'PRUEBA'/

de lo que se deduce que la forma general de la instrucción DATA es:

DATA variables/valores de las variables/,variables/valores de las variables/...

Cuando el valor a atribuir haya de considerarse constante o se quiera evitar el riesgo de modificaciones por parte de instrucciones de asignación erróneas se utilizará la instrucción PARAMETER en la forma:

PARAMETER (variable=valor)

que hace que la expresión "valor" sea tratada como una constante propiamente dicha a lo largo de todo el programa.

También se pueden definir constantes en función de otras constantes, siempre que estén definidas anteriormente por PARAMETER, para conservar valores más precisos. Por ejemplo:

PARAMETER (PI=3.14155926536, PIPOR3=3*PI,
TERCIO=1.0/3.0)

La constante definida podrá utilizarse en expresiones aritméticas o instrucciones DATA y podrá servir también para definir variables de tipo diferente al establecido por la letra inicial, a condición de que la instrucción IMPLICIT anticipe la declaración.

Expresiones aritméticas

Dentro de las instrucciones de asignación es posible definir expresiones aritméticas complejas, que se controlarán aplicando las reglas del álgebra tradicional.

Las operaciones admitidas son:

- elevación a una potencia (símbolo **)
- multiplicación (símbolo *) y división (símbolo /)
- suma (símbolo +) y resta (símbolo -)

indicadas en orden de prioridad. Ello significa que la expresión

I = 3 + 4 * 2

se resolverá como 3 + 8 y no como 7 * 2, al ser la multiplicación una operación con una prioridad mayor que la suma.

De cualquier modo, para evitar equívocos, será preferible utilizar los paréntesis, por cuanto que el Fortran les atribuye el máximo nivel de prioridad y, por consiguiente, los trata de forma correcta.

En efecto, la expresión

I = (2 + 3) * * 2

dará como resultado 25, por cuanto que la prioridad de la elevación a una potencia es superada por la correspondiente a los paréntesis.

Cuando en una expresión aparecen variables numéricas de tipo diverso, Fortran realiza directamente una transformación adecuada de los números para pasarlos a un tipo homogéneo, por lo general de tipo real. Ello es tanto más significativo cuando se considera que una variable entera, al aceptar un valor real, "trunca" la parte fraccionaria. Por consiguiente:

```
I = 4.5
```

se resolvería en I = 4

Ello significa que la expresión

```
I = 1
A= 3.0
J = 2
K= (I / A) + (J / A)
```

será valorada como 0 por cuanto que I/A proporcionará 0.333333 y J/A proporcionará 0.666666, pero su suma (0.999999) asignada a una variable entera será "truncada" en toda su parte fraccionaria.

Las cadenas

Las variables del tipo de carácter pueden sufrir muy pocas manipulaciones con respecto a las de tipo numérico. La obligación de definir la longitud de cada una de ellas hace inviable cualquier operación que no sea la de "ensamblaje" de varias variables de cadena. Por consiguiente, si queremos unir las variables que contienen los valores 'esta ', 'es una ' y 'cadena'. obtendremos:

```
    CHARACTER*7 A*5,B,C,D*19
    A= 'Esta '
    B = 'es una '
    C= 'Cadena.'
    D= A //B //C
    WRITE(*,100) D
100 FORMAT (1H ,A18)
    END
```

cuyo resultado será:

Esta es una cadena.

(posteriormente veremos cómo funcionan los comandos de impresión y formato).

Como se puede observar, para las variables de cadena se declaró una longitud tipo de 7 caracteres y cada variable de longitud diferente se declara con su correspondiente dimensión.

Además, se puede obtener una subcadena de una variable especificando cuál es el carácter a partir del cual debe iniciarse y dónde debe terminar la extracción de la nueva variable. Por ejemplo, a partir de la cadena 'ejemplo', será posible obtener 'empl', insertándola en la variable B, de este modo:

```
CHARACTER A*7,B*4
A= 'Ejemplo'
B = A(3:6)
```

Si no se indica uno de los números se entenderá que es el extremo correspondiente; así A(3:) proporcionará 'emplo' y A(:3) proporcionará 'eje'.

Con la misma sintaxis se puede modificar el valor de una cadena en alguna de sus partes; por ejemplo, aplicando a la variable vista antes la expresión

```
A(3:) ='ercito'
```

el contenido de A sea igual a 'Ejército'.

Puesto que la operación de asignación no es una expresión de igualdad, sino que se realiza mediante un mecanismo de transferencias sucesivas de valores entre diferentes celdas de memoria, no se podrá efectuar una asignación de una subcadena a otra que contenga alguno de sus caracteres, es decir, no será posible:

```
A(3:5) = A(5:6)
```

por cuanto que el carácter en la posición 5 aparece en ambos miembros de la expresión, mientras que sería lícita

```
A(3:4) = A(6:7)
```

que produciría en A, 'Ejloplo'.

Para la manipulación de las variables de cadena existen las funciones especiales siguientes:

LEN(A)
proporciona un valor entero igual a la longitud de la cadena;

ICHAR(A)
da como resultado un valor entero correspondiente al número de orden del carácter en la tabla de códigos del ordenador;

CHAR(I)
proporciona el carácter correspondiente, en la tabla de códigos utilizada por el ordenador, al número entero dado;

INDEX(A1,A2)
ofrece el valor entero correspondiente a la posición del primer carácter de la subcadena A2 en la cadena A1.

CAPITULO IV

FORTRAN: INSTRUCCIONES DE ENTRADA/SALIDA

Formato de las líneas

Un programa en Fortran es una lista de instrucciones que deben ejecutarse en el orden establecido por su secuencia.

Cada línea está constituida por 80 posiciones que pueden contener todos los caracteres del juego admitido por el ordenador. Las seis primeras columnas de cada línea están destinadas a contener etiquetas que han de anteponerse a las instrucciones para señalar pasos particulares del programa o disposiciones concretas. De la columna 7 a la 72 se incluirán las instrucciones efectivas, y en las columnas 73 a 80, los caracteres de identificación del programa.

El motivo de esta disposición ha de buscarse, desde el punto de vista histórico, en el empleo de las tarjetas Hollerith de 80 caracteres como soporte principal de las informaciones a introducir en el ordenador, de las cuales la disposición de las perforaciones rectangulares representaba a los caracteres existentes en la línea de programa.

En una línea de programa no puede indicarse más de una instrucción; un comando puede extenderse en varias líneas a condición de que en la sexta columna de las líneas sucesivas a la primera se registre un carácter diferente de 0 o del espacio, que tendrá el significado de indicar al compilador que la línea es una continuación de la anterior.

Otra posibilidad es la inserción de un asterisco o de una "C" en la primera columna, lo que supone que los caracteres de la línea son un comentario que será ignorado, por consiguiente, por

HOJA DE CODIFICACION FORTRAN

ICAI | Curso ______ | CODIGO DE USUARIO | CODIGO DE PROGRAMADOR | CODIGO DE TRABAJO | TIEMPO PREVISTO | LINEAS PREVISTAS

NOMBRE DEL PROGRAMADOR

PROGRAMA NUM. FECHA HOJA N.º DE — NOMBRE DEL PROGRAMA

C NUM. — SENTENCIA FORTRAN →

Figura 1.— Hoja de codificación típica en FORTRAN

el compilador. Es normal el uso de hojas de codificación similares a la reproducida en la figura 1.

Entrada/Salida

Dentro de un programa es frecuente la necesidad de adquirir datos de dispositivos externos (entrada) o de producirlos mediante envío a periféricos externos (salida). En efecto, se considera como entrada todo lo que la unidad central de proceso recibe de las unidades externas, tales como el teclado, el lector de cintas, las unidades de discos, etc., y como salida, todo lo que el ordenador envía al exterior (pantalla, unidad de discos, perforador de cintas, impresora, etc.).

Las instrucciones que desempeñan este cometido son READ (lectura) y WRITE (escritura), cuya sintaxis (igual para ambas) es:

```
READ (lista_de_opciones)var_1, var_2, ...
```

Esto especifica que de la unidad indicada dentro de "lista_de_opciones" deben extraerse los valores a atribuir, en el mismo orden de adquisición, a las variables var_1, var_2, etc.

En "lista_de_opciones" pueden especificarse diversas elecciones:

UNIT, FMT, REC, ERR, END, IOSTAT

UNIT — especifica la unidad periférica, entre las que están realmente conectadas, con la cual ha de efectuarse el diálogo. El empleo de un asterisco (*) comunica al compilador que se tiene la intención de extraer la información de la unidad estándar de entrada (normalmente el teclado). Para intercambiar datos con una unidad lógicamente no conectada se deberá proceder, mediante la instrucción OPEN, a la apertura de un canal de comunicación entre las dos unidades.

FMT — indica la etiqueta, dentro del propio programa, a continuación de la cual se especifica el formato de las variables tratadas. Se trata de una llamada concreta a la segunda parte de la instrucción de entrada/salida (vimos un ejemplo en el capítulo anterior).

REC — se utiliza con referencia a las unidades de discos, por cuanto que en los discos los datos están estructurados en ficheros que, a su vez, están subdividi-

dos en registros. REC especifica qué grupo de datos hay que procesar para obtener el valor requerido.

ERR representa la etiqueta, dentro del programa, que señala la instrucción a partir de la cual el programa se reiniciará en el caso de que la operación de lectura no se haya realizado de forma satisfactoria.

END representa la etiqueta, dentro del programa, que señala la instrucción a partir de la cual el programa se reiniciará cuando, al leer un fichero de disco, se encuentra en el mismo el final en el curso de la operación de lectura. El éxito de la operación de lectura puede averiguarse con facilidad, por cuanto que el sistema operativo restituye un valor indicativo.

IOSTAT especifica el nombre de la variable entera en la cual el sistema deberá depositar dicho valor indicativo.

La única opción que ha de especificarse de forma obligatoria es la relativa a la unidad periférica. El orden en el que las opciones deben proporcionarse no es vinculante.

Por consiguiente, el comando

```
READ (UNIT=*,FMT= 100, ERR= 999)
```

comunicará al compilador que se está a la espera de adquirir un dato a partir del teclado (unidad de entrada por defecto), que el formato del dato tecleado vendrá determinado por las indicaciones proporcionadas en la etiqueta 100 y que el programa, en caso de error en la entrada, se reiniciará a partir de la instrucción marcada con 999.

El formato en las operaciones de E/S

La parte de formato de la instrucción de entrada/salida tiene la sintaxis siguiente:

```
nºetiq. FORMAT(especificaciones_de formato)
```

en donde "nºetiq." representa la etiqueta llamada por la opción FMT y "especificaciones_de formato" contiene, en estricta correspondencia biunívoca con var_1, var_2, etc., las modalidades de conversión entre la representación interna de los datos, que se realiza dentro de la memoria, y la externa (establecida por las declaraciones implícitas o explícitas) que se encuentra en las unidades periféricas.

Por ello:

```
    READ (UNIT= *,FMT= 100, ERR= 999) I,J
100 FORMAT (2I4)
```

especificará que se deberán adquirir, a partir del teclado, los valores numéricos de dos variables enteras, a cada una de las cuales corresponderán 4 cifras.

En lo que respecta a la salida de los datos a los periféricos, la sintaxis de los comandos que han sido examinados es la misma. Sólo se modifica la instrucción READ, que se transforma en

```
WRITE
```

mientras que para el resto (opciones y descripciones de FORMAT) son válidas las mismas consideraciones hechas para la lectura.

La utilidad de las descripciones de FORMAT se pone de manifiesto en su plenitud cuando se consideran fichas o registros de datos, en los cuales la estructuración de las informaciones en columnas, es decir, con los datos alineados uno tras otro, hace necesaria una precisa subdivisión de las atribuciones.

En efecto, cuando se tenga que leer, por ejemplo, un registro constituido como:

```
ROJO_ _ _ _ _MARIO_ _ _ _ _ 35
```

en donde el símbolo "_" indica espacios en blanco, y se tenga que hacer con 3 variables diferentes, las instrucciones

```
    CHARACTER* 10 APELLI, NOMBRE
    INTEGER EDAD
    READ (UNIT= *, FMT= 100) APELLI,NOMBRE,EDAD
100 FORMAT (2A10, I3)
```

informarán al compilador de que las 10 primeras columnas del registro se refieren a la variable APELLI, que contendrá "ROJO_ _ _ _ _ ", que las columnas 11 a 20 corresponden a la variable NOMBRE, que contendrá "MARIO_ _ _ _ _ " y que las 3 últimas columnas corresponden a la variable entera (se declaró como tal) EDAD, que contendrá el valor "35".

Es oportuno precisar ahora cuáles son las especificaciones de formato utilizables en operaciones de escritura y de lectura:

nIx señala que "x" caracteres corresponde a una variable entera; por consiguiente, nIx indica que ese for-

mato corresponderá a "n" variables enteras. Si no está especificado el valor de "n", se supondrá 1.

nFx.y — indica que "x" caracteres corresponden a una variable real con "y" cifras decimales, para "n" variables. El número de cifras decimales se utiliza solamente si en los caracteres no está incluido ya un punto decimal.

nAx — "x" caracteres corresponden a una cadena.

A — han de tratarse tantos caracteres como hayan sido declarados en la definición de la variable indicada.

nX — deberán ignorarse "n" caracteres.

nH — indica que los "n" caracteres que siguen a H deberán considerarse como una cadena.

Tx — el carácter siguiente debe buscarse (o situarse) en la columna "x".

TLx — el carácter siguiente debe buscarse (o situarse) "x" columnas a la izquierda.

TRx — el carácter siguiente debe buscarse (o situarse) "x" columnas a la derecha.

De utilización más especial son las instrucciones siguientes:

/ — salta a la línea (o al registro) siguiente.

: — pone fin al formato cuando no se quieren describir todas las variables.

SP — imprime el signo "+" antes de los números positivos y el "–" antes de los números negativos.

SS — no imprime el signo "+" antes de los números positivos, pero sí el "–" antes de los negativos.

S — deja al procesador la elección sobre la indicación de "+"

kP — multiplica los datos por el factor de escala "k"

BN — ignora los espacios en los campos de entrada numérica.

BZ — trata como ceros los espacios en los campos de entrada numérica.

Ix.y — genera números enteros de, al menos, "y" cifras, completando, si es necesario, el número con ceros a la izquierda.

Fx.y — trata los valores reales con la indicación del exponente de 10; "y" son las cifras decimales y "x" el número de caracteres ocupados (incluidos signo, punto y exponente). Conviene que x>, y+7

Ex.yEz — trata los valores reales con "z" cifras en el exponente de 10.

Dx.y — trata los valores con doble precisión.

Gx.y — genera el formato de salida adecuado para el dato.

Gx.yEz — trata el formato de salida adecuado, con "z" cifras de exponente.

Es conveniente especificar que en la impresión el primer carácter de cada línea se interpreta como una señal de control sobre funciones de la impresora y no como carácter de salida. Los valores reconocidos son:

espacio — avance de una línea antes de la salida.

0 — avance de dos líneas antes de la salida.

1 — avance a la página siguiente antes de la salida.

\+ — ningún avance antes de la salida.

Particularidades y ejemplos

Existe otra instrucción para la salida, útil solamente en caso de datos muy simples por cuanto que no permite una indicación tan completa como WRITE y que se denomina PRINT.

La instrucción PRINT no suele hacer referencia a una lista de descriptores de formato, sino solamente a la unidad de salida estándar (normalmente la pantalla) y a las variables o a las cadenas indicadas a continuación.

Así:

```
A= 1.0
B = 2.0
PRINT*, 'He aquí a dos números reales', A,' y ',B
```

dará como resultado:

```
He aquí dos números reales 1.0 y 2.0
```

Por consiguiente, PRINT se utiliza sobre todo en la impresión de mensajes o en la verificación de los procedimientos.

Una particularidad de READ y de WRITE es que las indicaciones UNIT y FMT pueden omitirse si se presentan como primera y segunda opción de la lista. Por lo tanto:

```
READ (UNIT=1,FMT=200) A
READ (1,FMT=200) A
READ (1,200) A
```

son perfectamente equivalentes.

En lo que respecta a los formatos de entrada, al tener dos registros constituidos como sigue

```
___45___6745___23_____0
_2344_345550___24_____1
```

es conveniente examinar el resultado de diferentes especificaciones de lectura. Por ejemplo:

```
      CHARACTER*5 B
      READ (10, 100) A,I,B,J
  100 FORMAT (F5.2,3X,I4,A5,4X,I1)
```

producirá en A el valor 0.45, saltará 3 caracteres, insertará en I el valor 6745, en B el valor '   23 ', saltará 4 espacios e insertará en J el valor 0.

La modificación de una indicación puede producir grandes trastornos. Por ejemplo:

```
100 FORMAT (F5.2/3X,I4,A5,4X,I1)
```

producirá en A el valor 0.45, pasará al registro siguiente, saltará 3 caracteres, insertará en I el valor 4403, en B '45550', saltará 4 caracteres e insertará en J el valor 4. Todo ello sucede así porque el salto de registro lleva al comienzo del registro siguiente, en el cual la disposición de los datos es diferente. En particular, el valor 4403 se deriva del hecho de que Fortran trata como "ceros" todos los espacios encontrados en la lectura de valores numéricos. Muchas veces es preferible hacer que ignore los espacios encontrados en la lectura de los datos numéricos. Procediendo así hubiéramos obtenido el valor 443.

Este modo de actuar, adoptado por motivos de compatibilidad con las antiguas versiones del Fortran pueden crear algunos problemas al utilizar sistemas diferentes. Para subsanarlo y asegurarse de tratar la lectura de los datos numéricos como mejor conviene es oportuno utilizar, al comienzo de FORTMAT, las opciones BZ o BN que permiten, respectivamente, tratar los espacios

como "ceros" o ignorarlos siempre que se estén leyendo datos descritos por I,F,E,D,G.

Continuando el examen de las peculiaridades de las operaciones de entrada/salida se puede observar que, a veces, la utilización de la instrucción FORMAT (que al no ser una instrucción de ejecución, sino de declaración de valores, puede disponerse en cualquier lugar del programa), no resulta muy cómoda y para formatos simples hace pesada la escritura del programa.

En estos casos se puede sustituir, en la instrucción READ o WRITE, la opción FMT = por su valor efectivo.

En nuestro caso, podremos escribir:

```
READ (UNIT=1,FMT='(F5.2,3X,I4,A5,4X,I1)')
```

o, sin más, insertar la cadena FORMAT en una variable y utilizar esta última dentro de la línea READ, es decir:

```
FTO= '(F5.2,3X,I4,A5,4X,I1)'
READ (UNIT=1,FMT=FTO)
```

y obtener el mismo efecto.

También en este caso será posible omitir las descripciones de opción y escribir:

```
READ (1,'(F5.2,3X,I4,A5,4X,I1)')
```

Un problema relativamente frecuente es el de la indicación, dentro de la instrucción FORMAT, de un número de caracteres insuficiente para representar completamente el número requerido. Dicha circunstancia, que en la lectura produce un "corte" del valor, dará lugar en la salida a una línea de asteriscos. Así, al utilizar el formato F10.4 para describir el número -45322.6, la salida obtenida será ********** porque 10 los caracteres previstos no llegan a describir el signo (1 carácter), las 5 cifras antes del punto decimal, el propio punto decimal (1 carácter) y las 4 cifras de la parte fraccionaria (6000).

Por consiguiente, la razón que mueve a utilizar el formato "G" (G10.4) en estos casos es la posibilidad de delegar en el sistema la toma de decisión sobre el formato de salida adecuado.

Otro problema lo constituye el tratamiento de las cadenas.

Se vio con anterioridad que una variable de cadena se declara también en longitud para proporcionar información precisa al ordenador sobre la ocupación de memoria correspondiente; por ello, cuando se emplea una variable de cadena, se utiliza el descriptor de formato "A" para emplear el mismo número de carac-

teres usado en su declaración (h), mientras que con Ax es oportuna una mayor atención.

En efecto, en la fase de lectura si "x" es mayor que el número de caracteres "h" efectivo solamente se leerán los "h" caracteres de la derecha, mientras que si "x" es menor que "h" se añadirán h-x caracteres de espaciamiento al final de la cadena. Con más detalle, si un registro contiene

```
CADENAS
```

la operación

```
CHARACTER*5 CADENA
READ (1,'(A7)') CADENA
```

leerá 7 caracteres, pero insertará en CADENA el valor 'DENAS', mientras que

```
CHARACTER*7 CADENA
REAS (1,'(A7)') CADENA
```

leerá 5 caracteres y producirá, en CADENA, el valor 'CADEN'.

En caso de salida, con CADENA conteniendo 'CADENAS', la operación

```
CHARACTER*7 CADENA
WRITE (*,'(1H ,A9)') CADENA
```

producirá la impresión de

```
'  CADENAS'
```

mientras que:

```
CHARACTER*7 CADENA
WRITE (*,'(1H ,A5)') CADENA
```

producirá

```
'CADEN'
```

Observe el empleo de 1H como carácter de control a enviar como primer carácter de la línea de salida.

Unidades externas

Cada sistema comprende algunas unidades exteriores, tales como la representación visual, teclado, unidad de discos, impresora, etc., a las cuales se le asignan valores determinados de unidades.

Solamente las unidades de entrada y de salida definidas como estándar y, por consiguiente, identificadas por READ(UNIT= *) y WRITE(UNIT= *), se consideran conectadas al programa y siempre disponibles. Cuando se quieran utilizar las demás unidades, será preciso establecer una conexión lógica que se activa con el comando

```
OPEN (lista_de_opciones)
```

Las opciones admitidas son:

UNIT,FILE,STATUS,ACCESS,FORM,RECL,BLANK,ERR,IOSTAT

cuyo significado veremos después.

Recordemos que en los discos los datos se agrupan en registros que, a su vez, están reagrupados en ficheros e identificados por su nombre (fig. 2).

Los registros pueden ser de dos tipos, según que los datos se lleven como sus representaciones externas (lo que favorece el intercambio de los datos entre diversos sistemas), en cuyo caso toman el nombre de registros formateados (en inglés FORMATTED) o que se reproduzcan en el modo en el cual los conserva la me-

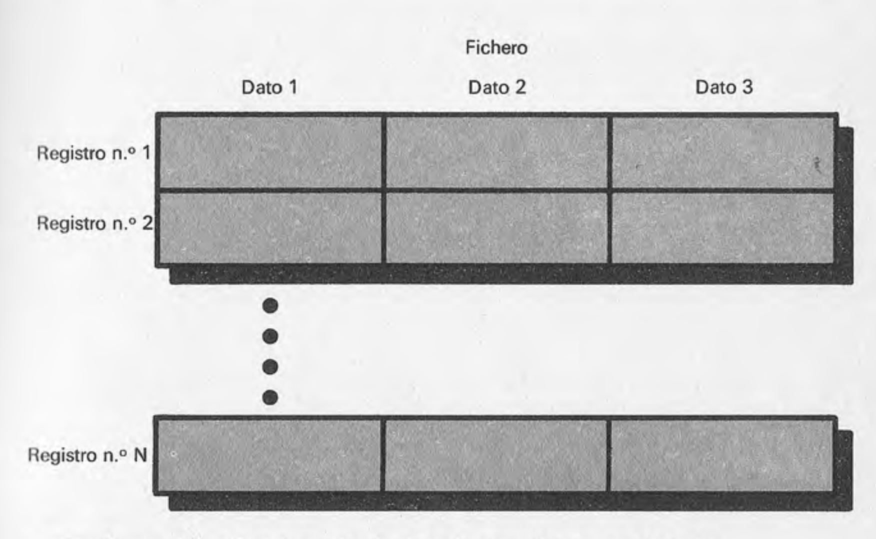


Figura 2.— Distribución de la información en un fichero.

moria, es decir, en código binario, en cuyo caso se definen como registros no formateados (en inglés UNFORMATTED).

Los registros formateados se controlan con instrucciones normales de READ y WRITE, en las cuales se especifica la opción FMT, que no debe ser declarada, por el contrario, en caso de registros no formateados.

Es conveniente cuando se utilicen registros no formateados que se tenga una idea precisa de cómo las distintas variables se conservarán dentro de la memoria, es decir, el número de bytes ocupado por cada una, con el objeto de valorar la longitud de cada registro que, en el caso de registros formateados, es simplemente la suma de los caracteres empleados en las descripciones de formato. Así:

```
READ (1) X, Y
```

proporciona solamente la indicación de leer dos valores reales, pero es necesario conocer las características del sistema para saber si los caracteres leídos serán 8, 12 u otro valor. Mientras que:

```
    READ (1,100) X, Y
100 FORMAT (2F10.3)
```

indica claramente que se tienen que leer 20 caracteres.

Otra diferencia consiste en el hecho de que los registros formateados pueden leerse o escribirse más de uno a la vez, lo que está prohibido en el tratamiento de los registros no formateados.

Opciones de OPEN

Profundicemos en el significado de las opciones de OPEN:

- UNIT tiene el mismo significado supuesto en READ y WRITE. Una oportunidad adicional la proporciona la posibilidad de especificar como UNIT una variable de tipo de carácter, que puede llegar a ser el instrumento para transferir informaciones entre ficheros de una manera rápida. Es la única opción obligatoria.
- FILE es el nombre del fichero exterior a emplear en la unidad conectada. Desde este momento, el fichero estará accesible al programa.
- STATUS estado del fichero, que puede ser: ya existente (OLD), nuevo (NEW), desconocido (UNKNOWN) o de tipo especial, útil para procesos internos, destinado a desaparecer después de la ejecución (SCRATCH).

- ACCESS es la modalidad de acceso al fichero. Puede ser secuencial (SEQUENTIAL), que se utiliza con la lectura y escritura de los datos uno tras otro, característico de los ficheros en cinta, o bien de acceso directo (DIRECT), típico de los ficheros en disco, en los cuales el fichero está subdividido en registros de la misma longitud, lo que posibilita el acceso directo a cualquier dato sin tener que leer o escribir todos los anteriores. En condiciones normales, si no se especifica ACCESS, el fichero se procesa de modo SECUENCIAL. La diferencia presupone un significado particular en cuanto que la escritura de un dato en un fichero secuencial destruye todos los datos siguientes, por lo que un fichero secuencial, en escritura, aumenta siempre su extensión, mientras que en los ficheros de acceso directo la escritura de un registro deja inalterados los valores de los datos anteriores y siguientes.
- FORM indica el tipo de registro y puede ser FORMATTED (FORMATEADO) o UNFORMATTED (NO FORMATEADO). A falta de especificación se suponen formateados los registros de los ficheros secuenciales y no formateados los ficheros de acceso DIRECTO.
- RECL indica la longitud del registro en el caso de ficheros de acceso directo. Ello significa que será necesario especificar la opción REC an READ y WRITE.
- BLANK tiene un significado análogo a las opciones BZ y BN en FORMAT. Se emplea cuando se quieren tratar como "ceros" los espacios dentro de los campos numéricos (ZERO) o se quieren ignorar (NULL).
- ERR indica la etiqueta que señala la instrucción a partir de la cual hay que volver a comenzar en caso de error.
- IOSTAT indica la variable en la cual el sistema insertará, al final de la operación, un valor dependiente del logro de la propia operación, valor que suele ser 0 en caso de resultados satisfactorios.

Un comando especial que se utiliza en los ficheros de acceso secuencial es ENDFILE; inserta en ellos una información particular que indica el final del fichero.

```
ENDFILE (UNIT=u, ERR=e, IOSTAT=I)
```

escribe el último registro en el fichero conectado a la unidad "u". Las otras opciones tienen el significado ordinario.

Después del empleo de ENDFILE ya no será posible escribir ni leer en el fichero correspondiente sin previo envío de un comando como

BACKSPACE (UNIT=u, ERR=e, IOSTAT=I)

que permite procesar el registro inmediatamente anterior o

REWIND (UNIT=u, ERR=e, IOSTAT=I)

que reposiciona el fichero al punto inicial, permitiendo así la lectura del primer registro.

Estado (INQUIRE) y cierre (CLOSE) de un fichero

Una vez agotadas las operaciones de escritura y lectura se puede "desconectar" la unidad mediante el comando

CLOSE (UNIT=u, STATUS=s, ERR=e, IOSTAT=I)

en el cual STATUS define la "suerte que le toca" al fichero después del cierre: se deberá mantener disponible (KEEP) o bien destruirse (DELETE).

Un comando especial es:

INQUIRE (lista_de_opciones)

que permite averiguar el estado y las características de un fichero cualquiera.

En efecto, en "lista_de_opciones" se especifican las variables en las cuales se insertarán los valores correspondientes a las diversas opciones y que podrán examinarse a continuación.

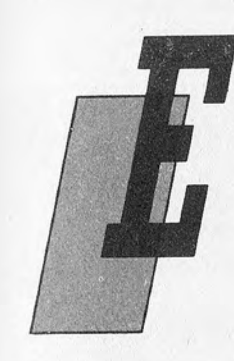
- UNIT (o, como alternativa, FILE especifica el objeto de la indagación.
- EXIST averigua la existencia del fichero, haciendo verdadera o falsa, según el resultado, la variable especificada.
- OPENED examina si el fichero está abierto, haciendo verdadera o falsa la variable especificada.
- NUMBER restituye un valor entero que indica la unidad conectada.
- NAMED examina si un fichero tiene nombre, haciendo verdadera o falsa la variable especificada.
- NAME restituye el nombre del fichero en la variable especificada.
- ACCESS proporciona en la variable especificada, dependiendo del tipo de registro, el tipo de acceso encontrado (SECUENCIAL o DIRECTO).

- FORM restituye FORMATTED (FORMATEADO) o UNFORMATTED (NO FORMATEADO).
- SEQUENTIAL, DIRECT, FORMATTED, UNFORMATTED restituyen SI o NO en las variables ligadas, dependiendo de la respuesta correspondiente.
- RECL proporciona, en la forma de un número entero, la longitud del registro.
- MAXREC proporciona, en forma de número entero, el número total de registros de fichero.
- NEXTREC proporciona el número de orden del registro siguiente al último registro leído o escrito.
- BLANK proporciona el NULO o CERO en la variable especificada.
- ERR e IOSTAT tienen el significado que asumieron ya en las otras instrucciones de operaciones con los ficheros.

CAPITULO V

FORTRAN: BUCLES DE CONTROL Y TOMAS DE DECISIONES

Bucles de control



n el curso de un programa es frecuente la repetición de un grupo de operaciones varias veces. Este problema particular, que obligaría a la escritura repetitiva de un gran número de pasos de programa iguales, se resuelve con la ayuda de la instrucción DO, cuya sintaxis es:

DO etiqueta, variable = val_inic, val_fin,Incremento

en donde "etiqueta" indica la línea que representa el final del bloque de instrucciones a repetir; "variable" corresponde a la variable que, en el ciclo de instrucciones, cambia su valor para tener en cuenta el número de ciclos a ejecutar; "val_inic." es el valor de partida de la variable; "val_fin" es su valor final, e "Incremento" es la magnitud en la que se incrementa después de cada ciclo.

Por consiguiente:

```
    DO 100,I=0,30,5
    WRITE(*,110)I
100 CONTINUE
110 FORMAT(1H ,I2)
    END
```

producirá:

```
 0
 5
10
15
20
25
30
```

por cuanto que la instrucción DO obligará al programa a ejecutar, de forma cíclica, el bloque de instrucciones que termina con la etiqueta 100. En la última línea de un bloque puede situarse realmente cualquier instrucción, pero se prefiere, con miras a la claridad, terminar con la instrucción CONTINUE, que no lleva a cabo ninguna acción. En el bucle se controla la variable I, que al comienzo tiene el valor 0 y que se incrementa de 5 en 5, tal como se pone de manifiesto por la impresión de los valores, hasta que toma el valor 30.

Hay que destacar el hecho de que un programa FORTRAN debe terminarse con la instrucción END.

El mecanismo interno de un bucle DO es muy sencillo:

a) a la variable se le asigna el valor inicial;
b) se ejecutan las instrucciones del bloque hasta la etiqueta indicada;
c) se suma, de forma algebraica, el valor del incremento al valor de la variable;
d) si la suma resultante es menor o igual que el valor final establecido el bucle vuelve a iniciarse a partir de la fase b);
e) si la suma resultante es mayor que el valor final indicado, el programa pasará a ejecutar la instrucción siguiente a la correspondiente etiqueta.

Esto trae como consecuencia que el valor que la variable tomará al final del bucle debe valorarse con atención, puesto que no corresponderá al valor final indicado, sino a un valor mayor.

En efecto, veamos qué ocurre al modificar el programa anterior según:

```
      DO 100,I=0,30,5
      WRITE(*,110)I
100   CONTINUE
      WRITE(*,120)I
110   FORMAT(1H,I2)
120   FORMAT(1H ,'el valor de salida del bucle es', I2)
      END
```


que producirá:

```
 0
 5
10
15
20
25
30
el valor de salida del bucle es 35.
```

El incremento puede representarse también por un valor negativo que al final del bucle será restado del valor inicial. Si no estuviera indicado, se sobreentenderá que el incremento vale 1.

Dentro de un bucle DO no es posible modificar el valor de la variable de control mientras sea posible su utilización. Esto es así porque la modificación de la variable alteraría la cuenta del número de bucles a ejecutar, que está almacenada en una posición de memoria no accesible al programa. Por ello tampoco es posible "entrar" en un bucle DO por una operación contenida en el bloque de instrucciones, por cuanto que el compilador no podría "comprender" los valores inicial, final y de incremento de la variable.

Un aspecto particular del bucle DO se presenta cuando la variable de control no es una variable entera, sino que se trata de una variable real. Ello produce efectos que han de valorarse con atención, por cuanto que la imprecisión con la cual un número se almacena dentro de la memoria hace necesarias ciertas precauciones. Puesto que el número de bucles a ejecutar es siempre un valor entero, cuando se obtiene a partir del cálculo de valores reales resulta de una truncación y no de un redondeo de la parte fraccionaria o decimal.

En efecto, el cálculo del número de bucles a ejecutar es el resultado de:

$$\frac{\text{val_final} = \text{val_inicial} + \text{incremento}}{\text{incremento}}$$

que, cuando implica valores reales, es un resultado aproximado. Para evitar, pues, que un valor como 9,999999 se considere 9 y no 10 como debería ser, es más prudente utilizar un valor final algo más elevado (solamente en las últimas cifras significativas) que el necesario, con el fin de compensar este efecto.

Por consiguiente, al encontrarse en una situación del tipo

```
DO 100, VAREAL = 0.01, 0.10, 0.01
```

sería mejor utilizar un valor de val_final igual a 0.100001 que dejaría, no obstante, intacta la naturaleza de los datos utilizados.

También es posible "anillar" varios bucles DO uno en el otro a modo de "cajas chinas". Es por tanto posible que una de las instrucciones de un bucle DO sea, a su vez, un bucle DO, y así sucesivamente. El único vínculo está representado por la obligación de que un bucle interno nunca podrá tener una instrucción fuera del bucle externo que lo contiene, aunque podrán terminar con la misma instrucción. Por consiguiente se admite

```
     DO 100,I=1,2
     DO 101,J=1,10
     ...
     ...
101  CONTINUE
100  CONTINUE
```

o bien

```
     DO 100,I=1,2
     DO 100,J=1,10
     ...
     ...
100  CONTINUE
```

Aunque el segundo caso podría engendrar ambigüedad y por esta razón se sigue prefiriendo la primera forma, mientras que no se admite

```
     DO 100,I=1,2
     DO 101,J=1,10
     ...
     ...
100  CONTINUE
101  CONTINUE
```

por cuanto que el bucle interior no está contenido por completo en el exterior.

Instrucciones de decisión

Las instrucciones de un programa suelen ejecutarse en el orden en el que están escritas, pero sucede a menudo que un grupo de instrucciones ha de ejecutarse solamente si se dan condiciones especiales o bien que se tienen que prever alteraciones

al normal desarrollo del programa que lleven a saltar instrucciones o a elegir alternativas entre ellas.

En Fortran existe la posibilidad de elegir ciertos criterios de examen de las condiciones, que deberán expresarse según la sintaxis del comando

```
IF  (condición_examinada) THEN
      bloque_de_instrucciones
ELSE
      bloque_de_instrucciones_alternativo
END IF
```

en el cual el grupo de instrucciones "bloque_de_instrucciones" se ejecutará si a la "condición_examinada" se da una respuesta positiva (en términos lógicos, si la condición es verdadera); de no ser así, es decir, si la condición fuera "falsa" (no se verifica) se ejecutará el "bloque_de_instrucciones_alternativo".

Para expresar una condición, se pueden utilizar seis operadores lógicos, que describen la relación existente entre dos términos. Así:

```
IF (A.GT.B) THEN
I = 0
ELSE
J = 0
END IF
```

producirá la ejecución de I=0 si A es mayor que B, mientras que se ejecutará J=0 si A es menor o igual que B.

No es obligatorio especificar ELSE y las instrucciones correspondientes, con el fin de "aligerar" el programa cuando no esté prevista la utilización de instrucciones alternativas.

A causa de las modalidades de representación interna de los números reales, la valoración de sus expresiones podría producir dificultades, sobre todo en el caso de comparaciones de igualdad. Por ello se prefiere, en aquellos casos en que las cifras significativas lo justifiquen, valorar no si dos términos son iguales, sino si su diferencia es aceptablemente pequeña. Así, mejor que

```
IF ((A. EQ. B)
```

se podría usar la expresión

```
IF (A–B). LT. 0.000001)
```

o cualquier otro número a nuestro criterio.

A menudo puede darse el caso de valoraciones complejas, en las cuales una condición, "verdadera" implica un posterior examen del estado de otra condición, y así sucesivamente.

En este caso, la expresión IF..THEN..ELSE se puede complicar al introducir niveles diferentes de examen. Así, se puede obtener:

```
IF (condición_1) THEN
  instrucciones_1_verdadera
ELSE IF (condición_2) THEN
        instrucciones_2_verdadera
        ELSE IF (condición3) THEN
              instrucciones_3_verdadera
              ELSE
              instrucciones_3_falsa
END IF
```

realizando un "árbol" de pruebas concatenadas.

También es posible el caso de que una instrucción tenga que ejecutarse si dos o más condiciones resultan verdaderas o si resulta verdadera al menos una entre un grupo de condiciones diferentes. Se utilizan para el primer caso dos o más expresiones. ".AND.":

```
IF ((I.EQ.0) .AND. (J.EQ.0)) THEN
        TOD = 0
ELSE
        NING = 0
END IF
```

producirá la ejecución de la instrucción TOD = 0 solamente si se verificaran I = 0 y J = 0. Si el resultado de la comparación fuera "falso", se ejecutará NING = 0.

Por el contrario, si se tiene

```
IF ((I.EQ.0) .OR. (J.EQ.0)) THEN
   TOD = 0
ELSE
   ING = 0
END IF
```

producirá la ejecución de la instrucción TOD = 0 si al menos una de las dos condiciones es verdadera.

El empleo de los paréntesis no es obligatorio, por cuanto que las expresiones de condición tienen una prioridad menor que las expresiones aritméticas, pero aumentan la claridad de la instrucción y evitan ambigüedades.

Junto a estas dos expresiones existen otras tres, menos utilizadas: .NOT., .EQV. y .NEQV., la primera de las cuales se emplea para invertir el sentido de una condición. Si

```
(A = 0)
```

es una expresión verdadera, entonces

```
.NOT.(A = 0)
```

será falsa.

.EQV. y .NEQV. se utilizan para comparar dos condiciones. Si ambas son del mismo tipo, es decir, si ambas son verdaderas o falsas, la expresión con .EQV. será verdadera, mientras que la expresión con .NEQV. será falsa.

Si fuesen de tipo opuesto (una verdadera y la otra falsa) como es el caso de que sea I = 0 y J = 1, se tendrá que:

```
IF ((I.EQ.0) .EQV. (J.EQ.0)) THEN
   TOD = 0
ELSE
   NING = 0
END IF
```

producirá la ejecución de NING = 0, al tratarse de dos expresiones de tipo diferente.

Sobre las expresiones lógicas de las que hemos hablado nos queda por decir que la escala de prioridad entre ellas impone que se ejecuten en el orden:

```
.NOT.
.AND.
.OR.
.EQV. y .NEQV.
```

aunque los paréntesis, como es habitual, pueden alterar el orden de ejecución.

Salto incondicional y condicional

La instrucción que logra modificar completamente el flujo del programa permitiendo saltar "de un lado para otro" entre los bloques de comandos adopta la forma

```
GOTO etiqueta
```

en donde "etiqueta" corresponde a la instrucción con la cual el programa deberá reiniciar la ejecución.

En efecto, GOTO es una instrucción que ha de utilizarse con moderación, por cuanto que hace fatigosa la lectura de un programa al romper su estructura. El salto continuo de una parte a otra sin adecuadas referencias o motivos suele ser indicativo de una construcción desordenada de la lógica del programa.

Una de las premisas de la programación estructurada, que cada vez más se va afirmando como indispensable para la construcción de buenos procedimientos, es precisamente limitar la existencia del GOTO a las construcciones lógicas internas y a instrucciones complejas, tales como IF..THEN y DO.

Existen otros modos de introducir un cambio del flujo de instrucciones dependiendo del valor de una variable de control.

La expresión:

```
GOTO (etiqueta_1 , etiqueta_2,...), expresión_entera
```

dirige la prosecución del programa a la "etiqueta_1" si "expresión_entera" da como resultado el valor de la posición de "etiqueta_1" en la lista (1), y así sucesivamente para todas las alternativas.

La expresión

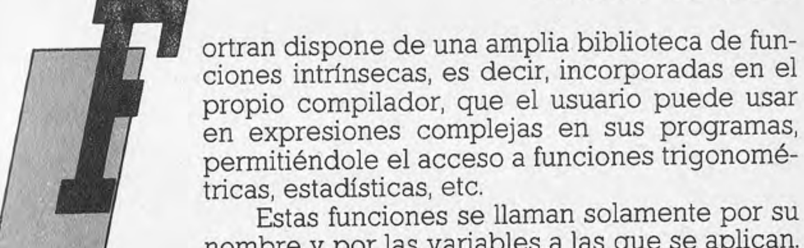
```
IF (expresión_aritmética) etiq_1 , etiq_2 , etiq_3
```

por el contrario, dirige la prosecución del programa a la "etiqueta_1" si el resultado de "expresión_aritmética" es menor que 0, a la "etiqueta_2" si es igual a 0 y a la "etiqueta_3" si es mayor que 0.

CAPITULO VI

FORTRAN: FUNCIONES, SUBRUTINAS Y MEMORIA

Funciones del usuario



Fortran dispone de una amplia biblioteca de funciones intrínsecas, es decir, incorporadas en el propio compilador, que el usuario puede usar en expresiones complejas en sus programas, permitiéndole el acceso a funciones trigonométricas, estadísticas, etc.

Estas funciones se llaman solamente por su nombre y por las variables a las que se aplican. Así, si queremos calcular la tangente de un ángulo, escribiremos

```
X = TAN (A)
```

para conseguir que en X se almacene el valor de la tangente del ángulo A expresado en radianes.

Muchas de estas funciones pueden aplicarse a variables numéricas de tipo diverso, que se reconocen por el tipo de argumento utilizado. Así, la función raíz cuadrada de un número puede aparecer como

A = SQRT(X)	si A y X son variables reales
A = SQRT(X) o A = DSQRT(X)	si A y X son variables de doble precisión
A = SQRT(X) o A = CSQRT(X)	si A y X son variables complejas

en donde DSQRT y CSQRT existen por compatibilidad con las antiguas versiones del Fortran, en las cuales las funciones aplicadas a tipos de variables diferentes tenían nombres diferentes.

En el apéndice se da una lista de las especificaciones de las diversas funciones intrínsecas y de los tipos de variables que proporcionan.

Además de ello, en Fortran está permitida, por parte del usuario, la creación de funciones que por su modo de operar, de utilizarse y llamarse son perfectamente similares a las descritas y que llegan a ser, en el programa, unas variables propiamente dichas.

Así, si hacemos referencia a las soluciones de las ecuaciones de segundo grado ($ax^2 + bx + c = 0$)

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

y recordamos que para valorar si las soluciones son reales se examina solamente si el término

$b^2 - 4ac$

es positivo, se podría escribir un programa que realizara esta prueba. Para hacerlo de una manera generalizada, conviene que la función se describa y utilice en un subprograma externo al programa principal. Esto se llevará a cabo mediante la creación de un conjunto de instrucciones que no formará parte del programa principal, sino que, bajo el nombre de la propia función, se añadirá al mismo.

La sintaxis a respetar, que prevé la declaración del tipo de la función, será en este caso:

```
Programa principal
REAL A,B,C,X,DISCR
X = DISCR (A,B,C)
END
```

que permitirá al compilador reconocer que DISCR es una función creada por el usuario (por cuanto que no está incluida en la lista de las funciones intrínsecas) y que se trata de variables reales.

Al final del programa, se añadirá un subprograma con la disposición siguiente:

```
REAL FUNCTION DISCR (X,Y,Z)
REAL X,Y,Z
DISCR = Y * y - 4.0 * X * Z
END
```


La existencia de un conjunto de instrucciones en el que aparece la palabra FUNCTION, precedida por su categoría y seguida por la lista de las variables que contribuye a obtener el valor buscado, permite al compilador organizar la ejecución del programa de modo que se desplace el control al subprograma FUNCTION llamado por el programa principal y se restituya, en la variable nombrada, el valor resultante.

Ha de ponerse de manifiesto que los nombres de los parámetros que aparecen después del nombre de FUNCTION no tienen ninguna importancia (una vez compilado, el subprograma hará referencia exclusivamente a las direcciones), siempre que exista una correspondencia plena entre los parámetros internos a la FUNCION (FUNCTION) y la lista de los utilizados para su llamada. En este caso, A corresponderá a X, B a Y y C a Z.

Es evidente que, dentro del subprograma FUNCTION, deberá aparecer la instrucción de asignación a la variable que le da nombre (en este caso, DISCR = ...).

Existe también una manera diferente de crear una función, esta vez interna al programa, que permite evitar el recurso a subprogramas siempre que la función sea definible mediante una sola expresión. Por ejemplo, la expresión

```
CHARACTER A
TEST (A) = A.GT.'A'.AND.A.LT.'Z'
```

define, de modo rápido, una expresión de la variable de cadena A susceptible de utilización repetida. El requisito exigido es que la definición de una función similar debe encontrarse antes de cada instrucción ejecutada.

Subrutinas

Un modo común de organizar un programa que requiera la ejecución repetida de un conjunto de instrucciones consiste en colocar estas instrucciones en una unidad de programa autónoma y hacer referencia a ella solamente cuando sea preciso.

Este conjunto de instrucciones toma el nombre de SUBRUTINA y tiene una naturaleza muy similar a la de FUNCTION, de la que se habló anteriormente.

La principal diferencia consiste en el hecho de que en el programa principal la lista de los parámetros que deberán encontrar correspondencia en los parámetros generales del subprograma va precedida por CALL. Así, si queremos crear un programa en el que se repita varias veces una instrucción de impresión, podremos escribir:

```
PROGRAM PRINC
CHARACTER*80 CADENA
CADENA = 'Primera cadena a imprimir'
CALL IMPRESION (CADENA)
CADENA = 'Segunda cadena a imprimir'
CALL IMPRESION (CADENA)
END
```

que deberá prever un subprograma del tipo

```
    SUBROUTINE IMPRESION (A)
    CHARACTER*80 A
    WRITE (*, 100)A
100 FORMAT (1H ,A80)
    RETURN
    END
```

en donde los parámetros locales realizan la correspondencia en número, tipo, longitud y orden con los de llamada, y la instrucción RETURN comunica al compilador que la ejecución deberá volver a iniciarse, en el programa principal, desde la instrucción sucesiva a la que había llamado a la subrutina.

Para distinguir los programas principales de las SUBRUTINAS y de las FUNCIONES (FUNCTION), se utiliza como primera instrucción PROGRAM seguida por el nombre identificativo del programa (6 caracteres).

A propósito de los parámetros locales de las subrutinas es necesario destacar que, en caso de variables de cadena, podrían surgir problemas con la declaración de variables de longitud diferente de la de los correspondientes parámetros en el programa principal, que son, en definitiva, los de utilización efectiva. Se utilizará en este caso la declaración genérica

```
CHARACTER * (*)
```

que asegura la adaptación automática del parámetro local al general. Ello sirve también cuando se tengan que declarar vectores de los cuales el dimensionamiento sea genérico, por lo que será conveniente, en el caso de un vector real de nombre A, la forma

```
REAL A (*)
```

dentro de la subrutina. Del modo en el que se almacenan los vectores en la memoria se deduce que, en caso de vectores multidimensionales, solamente la última dimensión podrá definirse por un asterisco.

Con el asterisco se pueden indicar también direcciones alternativas a partir de las cuales deberá reiniciarse la ejecución al retorno de la subrutina. La elección dependerá del valor adoptado por una expresión de números enteros dentro de la subrutina.

Así:

```
PROGRAM GUIA
...
CALL VARIAS (A,B,*10,*20,*30)
...
10 ...
...
20 ...
...
30 ...
...
END
y
SUBROUTINE VARIAS (X,Y,*,*,*)
...
...
RETURN iespr
END
```

Al retorno de VARIAS el valor de "iespr" determinará a partir de cual de las etiquetas indicadas como alternativa (10,20,30) el programa reiniciará la ejecución.

Cuando en la lista de los parámetros formales que una unidad de programa pasa a una subrutina están incluidas subrutinas o funciones, será necesario que se declare dicha circunstancia mediante las expresiones

```
EXTERNAL proc_1,proc_2,...
```

que es válido para los procedimientos externos y las funciones del usuario, e

```
INTRINSIC fun_1, fun_2,...
```

que sirve para las funciones intrínsecas, lo que es necesario cuando se tengan que utilizar funciones creadas por el usuario con el mismo nombre. Así si, por ejemplo, un programa principal llama a una primera subrutina pasándole como parámetro el nombre de otra subrutina y esta primera, a su vez, llama a una segunda que emplea también una función intrínseca como parámetro, tendremos:

```
PROGRAM PRINC
EXTERNAL PRIMERA
INTRINSIC SIN
...
CALL SUB_1 (A,SIN,PRIMERA)
...
END


SUBROUTINE SUB_1 (X,Y,COMUN)
EXTERNAL COMUN
...
CALL SUB_2 (B, COMUN)
...
RETURN
END


SUBROUTINE SUB_2 (Z,COMUN)
...
RETURN
END
```

en donde la declaración EXTERNAL se utiliza solamente cuando el procedimiento o la función entran a formar parte de la lista de los parámetros. Este es el motivo por el que en "SUB_2" COMUN no se declara como EXTERNAL, al no transferirse a otras unidades, mientras que si se hace esto en "SUB_1", que la utiliza como parámetro de llamada de "SUB_2".

Además, si no se hubiera declarado intrínseca (INTRINSIC) la función SIN, el compilador hubiera considerado SIN, dentro de la lista de los parámetros de llamada de "SUB_1", como una variable real de nombre SIN y no como la función intrínseca análoga.

Con mucha frecuencia, algunas subrutinas o funciones pueden reunirse para obtener provecho de las instrucciones comunes. Se utiliza, para diferenciarlas, la instrucción ENTRY, que permite llamar no a toda la subrutina, sino solamente una parte de ella, concretamente al bloque que sigue a la línea de definición de ENTRY.

Así:

```
SUBROUTINE PRUEBA (A,B,C)
REAL A,B,C
A= 2 * A
B = 4 * B
```



```
ENTRY PRUEBA1(A,B,C)
C = A + B
RETURN
END
```

permitirá el acceso a las dos subrutinas PRUEBA y PRUEBA1 en puntos diferentes. El funcionamiento de los subprogramas hace que las variables locales utilizadas no tengan significado en el ámbito del programa principal y que, en lugar de ello, su valor sea indefinido, una vez que las active una llamada de programa. No obstante, existen casos en los que resulta de utilidad que una subrutina o una función almacenen el valor tomado por algunas o todas las variables en el proceso anterior. Ello se consigue insertando, dentro del conjunto de las instrucciones y antes de cada instrucción DATA, la expresión

```
SAVE
```

que realiza el almacenamiento de todas las variables y vectores, poniéndolos a disposición de la sucesiva CALL,

```
SAVE var_1, var_2,..., vect_1 , vect_2,...
```

salvaguarda solamente los valores de las variables indicadas.

Memoria compartida

Haciendo referencia al proceso de compilación es útil recordar que el Fortran, para conservar "trazas" de todas las variables de un programa, a las que hace referencia mediante direcciones de posiciones de memoria, crea una especie de registro al que consulta para buscar rápidamente la información deseada.

Para aumentar la eficacia de este mecanismo da la posibilidad de identificar zonas de memoria en las cuales insertar valores que serán accesibles desde diferentes unidades de programa. Esto significa que las variables depositarias de dichos valores, una vez indicadas como comunes, podrán ser vehículo de transferencia de datos entre un programa y un subprograma, varios subprogramas o varias funciones.

La instrucción que realiza esta especificación tiene como sintaxis

```
COMMON /nombre_bloque_1/lista_variables_1 , /nombre_bloque_2/...
```

Por consiguiente, al introducir

COMMON /COM_1/A,B(1:20)

en un programa hará que a las posiciones correspondientes a las 21 variables indicadas se pueda hacer referencia mediante el nombre COM_1. Así, en otra sección será posible introducir una instrucción similar, en la cual se especificarán las variables que vayan a "apropiarse" de dichos valores, en el mismo orden. Es decir:

COMMON /COM_1/C,D(1:20)

hará que C esté en correspondencia con el valor que antes se relacionaba con A y que los elementos del vector D correspondan a los elementos del vector B.

Una importante observación sobre los tipos de variables que el Fortran utiliza se refiere a que los bloques COMMON no siempre pueden contener variables de tipo de carácter junto con los de tipo numérico, lo que obligaría en esos casos a declarar varios bloques COMMON.

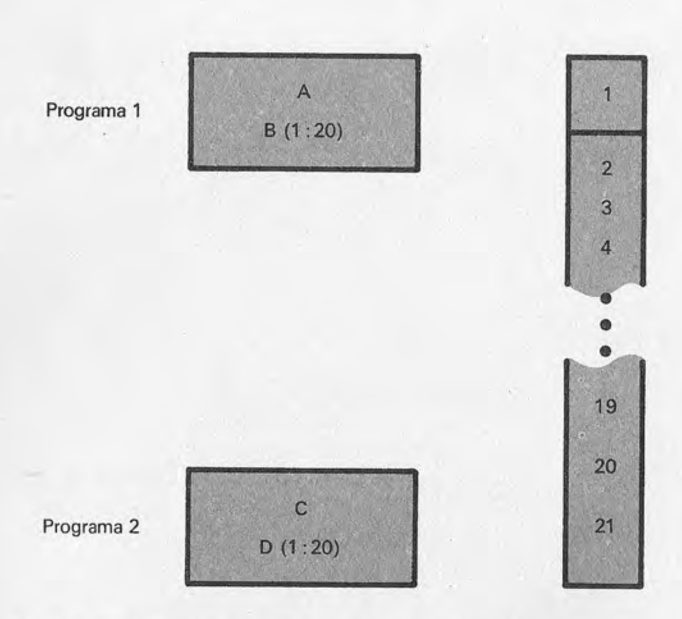


Figura 1.— Bloque COMMON para compartir memoria.

La declaración de un bloque puede hacerse de una sola vez o mediante varias referencias. Así, las expresiones

COMMON /BLOQ_1/A,B(10)
COMMON /BLOQ_2/F,G,/BLOQ_1/C(8)

equivalen para los fines del BLOQ_1, a

COMMON /BLOQ_1/A,B(10) , C(8)

Los bloques COMMON pueden ser también sin nombre, en cuyo caso asumen una importancia especial. En cada programa se admite un único bloque COMMON sin nombre, que hace de memoria "global" propiamente dicha del programa, mientras que los bloques etiquetados tienen una utilización limitada a aquellos casos en los que sus informaciones hayan de compartirse solamente por algunas unidades.

Las dimensiones de los bloques con nombres han de ser iguales en todas las unidades de programa que los utilizan, lo que no es necesario con el bloque sin nombre, al cual cualquier subprograma puede hacer referencia exclusivamente por la parte deseada. Así, si en un programa se tuviera

COMMON A,B(10), C(8)

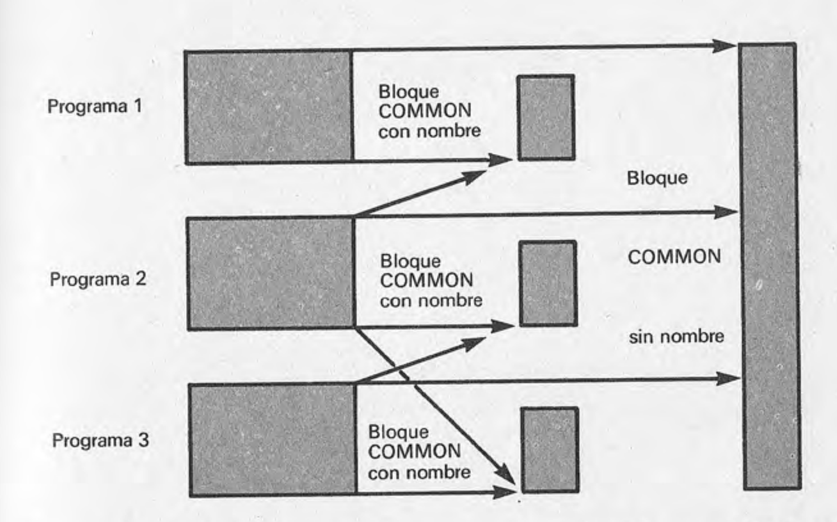


Figura 2.— Compartiendo memoria a través de bloques COMMON con o sin nombre.

en otro subprograma se podría encontrar

COMMON F

si interesara solamente la primera variable o, incluso COMMON G(S) para tomar los cinco primeros valores.

Otra diferencia entre los dos tipos de bloques está constituida por el valor inicial tomado por las variables que los constituyen. Los bloques sin nombre no están inicializados, mientras que para los segundos existe un subprograma específico (BLOCK DATA) dedicado a este cometido.

Por consiguiente, todas las variables contenidas en bloques etiquetados encontrarán su definición dentro de instrucciones DATA contenidas en una unidad como

```
BLOCK DATA nombre_bloque
...
DATA...
DATA...
...
END
```

Pero la diferencia principal entre los dos tipos de bloques COMMON consiste en el hecho de que las variables contenidas en los bloques sin nombre mantienen su valor aun cuando no se les haga referencia explícita y directa, mientras que si un bloque etiquetado no se describe en alguna subrutina, los valores de sus variables se perderán en el momento de la ejecución de dicha subrutina. El único modo de conservar los valores de los bloques etiquetados es introducir el nombre del bloque en una instrucción SAVE, con el fin de obligar al programa a conservar sus valores.

Así

```
SAVE var_1,...,/nombre_bloque/ ,...
```

serviría para la conservación de los valores de "nombre_bloque".

No obstante, se prefiere declarar en el programa principal como variables pertenecientes al bloque sin nombre, aunque no se las utilice, todas las variables que hayan de tener un empleo frecuente, con el fin de conservar siempre su valor.

Cuando la memoria disponible no sea muy amplia, o para emplear la posibilidad de referirse al mismo valor con varios nombres de variables, se utiliza la instrucción:

```
EQUIVALENCE (lista_variables_1) , (lista_variables_2) ,...
```


que informa al compilador que todas las variables (o elementos) de la lista 1 comienzan en la misma posición de memoria, que todas las variables (o elementos) de la lista 2 hacen lo propio, y así sucesivamente. Esto significa que:

```
EQUIVALENCE (A(24) , C(8) , P)
```

producirá el efecto mostrado en la figura 3.

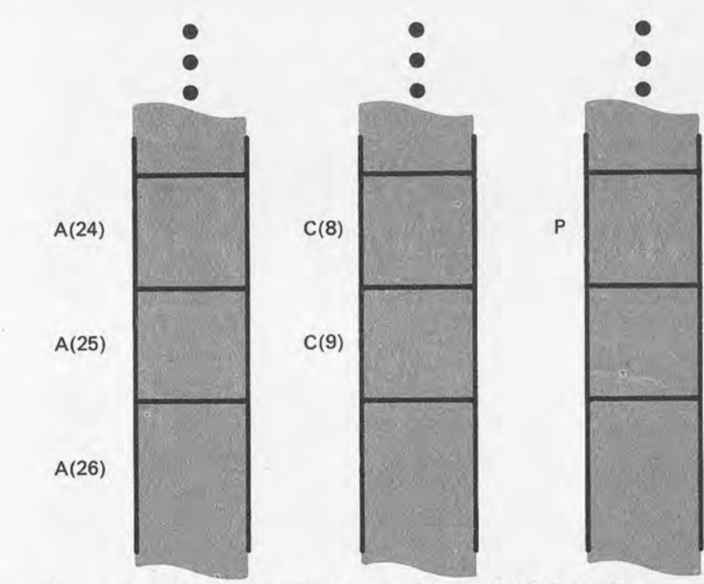


Figura 3.— Efecto de una instrucción EQUIVALENCE

más eficaz que una instrucción COMMON empleada con las respectivas listas de variables correspondientes.

En efecto, este método, en lugar de cálculos complicados para valorar la posición de cada variable en la lista COMMON, realiza la simple especificación de las variables inmediatamente correspondientes. Es evidente que una distribución similar de la memoria trae consigo que las celdas próximas sean ocupadas por los elementos correspondientes de los vectores, si los hubiere; así A(25) ocupará la misma posición que C(9) y lo mismo sucesivamente.

CAPITULO VII

COBOL: ESTRUCTURA



Debido a las características especiales del lenguaje COBOL resulta necesaria una visión general de la estructura que debe tener cualquier programa escrito siguiendo las reglas de su sintaxis.

Un programa COBOL está dividido en cuatro partes, llamadas DIVISIONES, cada una de ellas con cometidos y significados completamente distintos:

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION

Cada instrucción debe estar, según la función que desempeñe, dentro de una de estas cuatro partes. A su vez, las "divisiones" comprenden varias "secciones" y estas últimas varios "párrafos".

- IDENTIFICATION DIVISION (DIVISION DE IDENTIFICACION); incluye todas las instrucciones que sirven para describir el nombre del programa, la fecha de creación, el autor, etc., es decir, las informaciones que constituyen la "etiqueta" del programa.
- ENVIRONMENT DIVISION (DIVISION DEL ENTORNO); en él están las instrucciones relativas al entorno exterior al programa y, por consiguiente, el nombre del ordenador en el que está cargado y los nombres y características de los dispositivos exteriores y de los ficheros que se utilizarán por el programa.

- DATA DIVISION (DIVISION DE DATOS); incluye las definiciones de las variables que serán utilizadas en el programa y su estructuración.
- PROCEDURE DIVISION (DIVISION DE PROCEDIMIENTO): dispone de las instrucciones efectivas de desplazamiento de los datos declarados en las secciones anteriores.

Las divisiones están constituidas como muestra la figura 1.

IDENTIFICATION DIVISION

PROGRAM-ID	denominación del programa.
AUTHOR	nombre del autor.
INSTALLATION	instalación.
DATE-WRITTEN	fecha de creación del programa.
DATE-COMPILED	fecha de compilación.
SECURITY	comentario.

ENVIRONMENT DIVISION

	CONFIGURATION SECTION
SOURCE-COMPUTER	indica el ordenador en el que se compiló el programa.
OBJECT-COMPUTER	indica el ordenador en el que deberá trabajar el programa.
SPECIAL-NAMES	indica las denominaciones especiales utilizadas por el compilador.
	INPUT-OUTPUT-SECTION
FILE-CONTROL	asocia los nombres de los ficheros utilizados en el programa con los externos.
I-O-CONTROL	define las técnicas de gestión especiales de los datos.

DATA DIVISION

FILE SECTION	describe la estructura de los ficheros.
	descripción de los ficheros
WORKING-STORAGE SECTION	describe los datos.
LINKAGE SECTION	describe los datos compartidos con otros programas.

PROCEDURE DIVISION

secciones y párrafos de instrucciones.


Figura 1.— Constitución de las cuatro divisiones de un programa en COBOL.

Líneas de programa

Una línea de un programa COBOL está constituida por 80 columnas, aunque se utilicen solamente 72.

Las seis primeras están destinadas a contener el número de secuencia de la línea, que se utilizará en muy raras ocasiones.

La columna 7 contiene caracteres cuyo significado es identificar destinos particulares de la línea. Un carácter de espacio en la 7.ª columna identifica una línea normal de programa. Un asterisco indica que la línea contiene exclusivamente comentarios; por ello será ignorada por el compilador. Un guión "-" indica que la línea ha de considerarse continuación de la línea anterior. Una letra "D" indica que la línea se utiliza exclusivamente para correcciones. Una barra "/" indica que la línea es de comentario; la hoja se deslizará hasta el comienzo de la página siguiente (salto de página) antes de la impresión de la línea.

Las columnas 8 a 11 son denominadas "Área A". En esta zona deben colocarse los primeros caracteres de las denominaciones de las divisiones, de las secciones y de los párrafos, así como los principales indicadores de nivel de las diversas categorías de datos (FD, 01 y 77).

Las columnas 12 a 72 se definen como "Área B". En estas columnas se incluirán las definiciones de los datos de nivel inferior a FD, 01 y 77. En condiciones normales, en esta área están contenidas todas las instrucciones efectivas.

Las columnas 73 a 80, tal como para el Fortran, están destinadas a identificación del programa.

Un programa COBOL podrá contener, pues, líneas similares a:

```
000010  IMPRESION
000450*   ESTA ES UNA LINEA DE COMENTARIO
        01     ARTICULO DE ALMACEN.
               MOVE A TO B.
               03  SUB-CAMPO-DE-A          PIC 9(8).
        PROC-LOOP-TEST.
```

Se consideran caracteres de separación el espacio, la coma y el punto y coma. Dichos caracteres pueden separar también las diversas opciones de un comando, mientras que cada instrucción debe completarse con un punto.

Normalmente los programadores hacen uso de hojas de codificación como la reproducida en la figura 2.

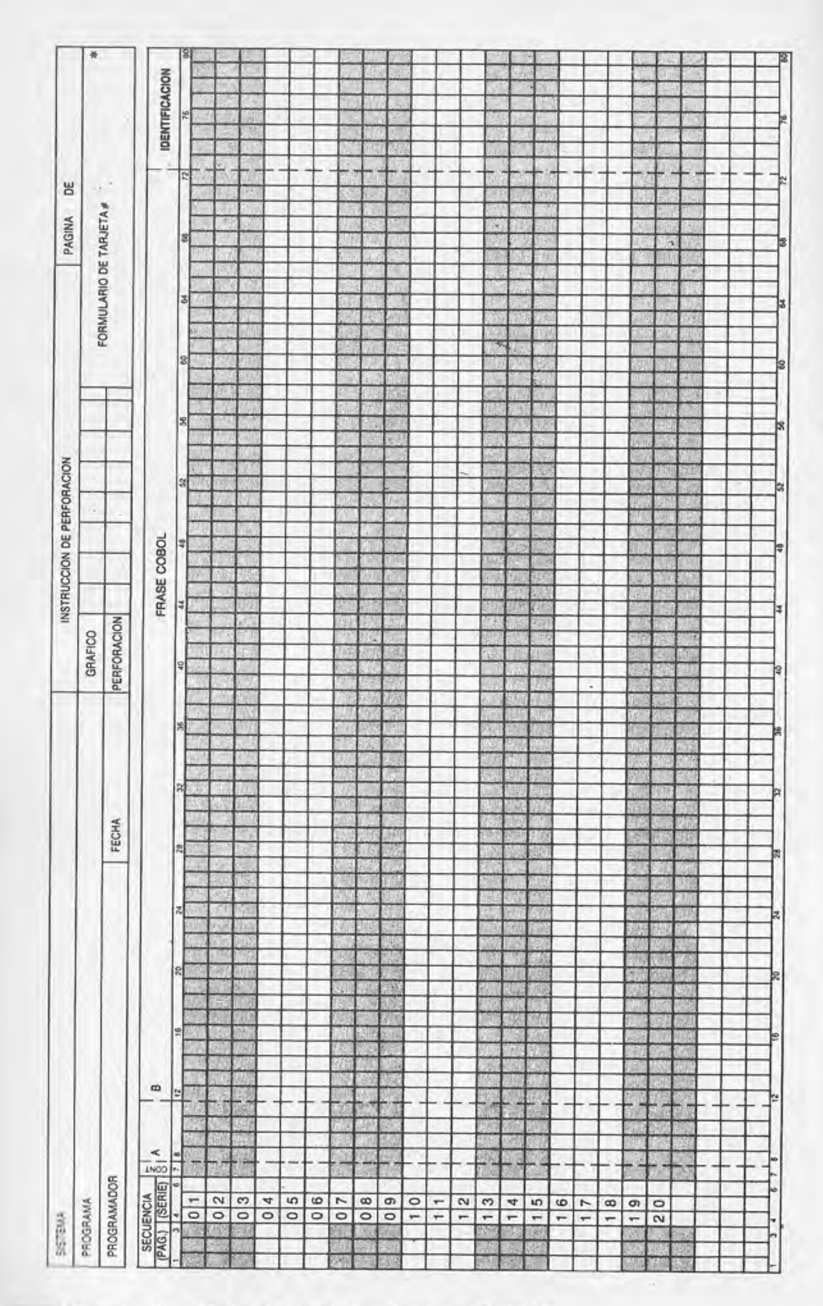


Figura 2.— Hoja de codificación en COBOL.

Los nombres en COBOL

El COBOL, por su propia naturaleza de lenguaje orientado al usuario, por su autodocumentación y por la propia sintaxis de los comandos, hace un gran uso de los nombres.

Los nombres, es decir las secuencias de caracteres alfabéticos y numéricos (incluido el guión "-"), pueden identificar prácticamente todas las informaciones procesables en un programa. En efecto, mediante los nombres se representan datos, constantes figurativas, ficheros, registros, índices, claves de acceso a los datos, partes de programa, dispositivos y variables.

La longitud de cada nombre puede extenderse hasta 30 caracteres, no pueden contener espacios dentro ni empezar o acabar en guión, y han de contener al menos un carácter alfabético, a no ser que se trate de nombres de párrafos o secciones. Serán, pues, nombres válidos:

0001 NOMINATIVO DESTINO-1 PROGR-NOV-85

Las constantes figurativas son:

ZERO, ZEROS o (ZEROES): indica el valor 0 o tantos valores "0" como sean los caracteres de la variable a los que se aplica.

SPACE o **SPACES**: indica el valor "espacio" o tantos valores "espacio" como sean los caracteres de la variable a los que se aplica.

QUOTE o **QUOTES**: indica comillas simples o dobles.

HIGH-VALUE o **HIGH-VALUES**: indica el carácter o los caracteres que tienen la posición más alta en el conjunto de caracteres del ordenador.

LOW-VALUE o **LOW-VALUES**: indica el carácter o los caracteres que tienen la posición más baja en el conjunto de caracteres del ordenador.

ALL: seguido por una cadena de uno o más caracteres indica que la variable a la que se aplica está constituida toda ella por los caracteres de la cadena. El valor resultante en la variable comprenderá un número de caracteres igual a su dimensión.

Variables

Las variables pueden representar valores de dos tipos: numérico y no numérico. Las de tipo numérico están constituidas por secuencias de cifras, un signo y un punto decimal (si fueran necesarios). Las de tipo no numérico están caracterizadas por un conjunto de caracteres encerrado entre dos comillas.

Como ejemplos para ambos tipos podemos considerar:

Tipo numérico	Tipo no numérico
324	"He aquí una cadena"
+0.556	"!"

Hay que señalar que para representar las comillas dentro de una variable de tipo no numérico se usan un par de comillas; así, "vinculante" se representará por:

""""vinculante""""

En realidad, cada variable, en el momento en que se declara, proporciona al compilador numerosas informaciones sobre su representación, por cuanto que su declaración puede ir acompañada por la definición PICTURE (o PIC) y por otras instrucciones que ilustran, de manera detallada, todas sus características; si es o no numérica, por cuántos caracteres está constituida, si debe considerarse el signo en su interior, si su representación interna es de tipo especial, etc.

Cada variable de tipo numérico puede comprender hasta 18 cifras, mientras que una variable no numérica puede comprender un número de caracteres que suele depender del ordenador y del sistema operativo en el que está instalado el lenguaje (desde 1 a 128, desde 1 a 2047, etc.).

Identification division (División de identificación)

La división inicial es necesaria solamente para los fines de documentación del programa.

En efecto, son obligatorias solamente las instrucciones

```
IDENTIFICATION DIVISION.
PROGRAM-ID.nombre_del_programa
```

mientras que son opcionales y, si están presentes, tiene que introducirse en el orden presentado:

```
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILATION.
SECURITY.
```


Environment division (División del entorno)

En esta división solamente son instrucciones obligatorias:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.nombre_del_ordenador
OBJECT-COMPUTER.nombre_del_ordenador
```

En el ámbito de OBJECT-COMPUTER pueden encontrarse también declaraciones facultativas, a modo de comentarios, como:

MEMORY[SIZE] n WORD/CHARACTERS/MODULES

que indica la cantidad de memoria expresada en la unidad de medida especificada

[PROGRAM COLLATING] SEQUENCE [IS].nombre_de_la_secuencia

que especifica la tabla de símbolos adoptada, en el caso de que se quieran modificar los valores correspondientes a HIGH-VALUE y LOW-VALUE, y

SEGMENT-LIMIT IS.número_segmento

que indica el máximo número de segmento utilizado (de 1 a 49). En lo sucesivo se utilizarán siempre los corchetes ([ ]) para indicar las palabras que, en el ámbito de una instrucción, pueden estar o no presentes sin modificar su sentido, mientras que las posibles alternativas irán precedidas por la barra "/" (ejemplo ON/OFF).

Por el contrario, son instrucciones facultativas:

```
SPECIAL-NAMES.
    SWITCH (desde 0 a 7) IS nombre_usado_en_el_programa
    ON/OFF[STATUS]IS.nombre_condición_1,...
    nombre_tabla_caracteres      IS STANDARD-1/NATIVE.
    CURRENCY[SIGN]IS.carácter
    DECIMAL-POINT IS COMMA
```

con las cuales se establece la correlación de nombre_usado_en_el_programa con el dispositivo exterior equivalente (SWITCH).

Se puede averiguar su estado mediante nombre_condición_1

(ON/OFF), que es una instrucción que se suele utilizar para sustituir nombres mnemónicos por instrucciones de control.

Se puede elegir la tabla de los caracteres a utilizar (que está en correlación con los valores de HIGH-VALUE y LOW-VALUE) entre las alternativas indicadas.

Se puede indicar qué carácter, en la declaración de la variable, deberá representar el símbolo de la moneda.

Se puede invertir el empleo del punto decimal con el de la coma por cuanto que, en los países de lengua inglesa, las cifras decimales están separadas por un punto, mientras que la coma se emplea para separar los millares, los millones, etc.

En ENVIRONMENT DIVISION, la sección que asocia los ficheros externos a los nombres que los representan en el programa es

INPUT-OUTPUT-SECTION.

En COBOL, los ficheros pueden ser de tipo diverso y procesarse de varios modos, por lo que existen diversas formas sintácticas para el párrafo

FILE-CONTROL.

según que los ficheros estén organizados de manera secuencial, relativa o con índice.

En un fichero secuencial los registros están almacenados en posiciones contiguas, uno tras otro, y para el tratamiento de uno de ellos será necesario procesar todos los anteriores. Por este motivo se trata de un tipo de organización adecuada para ficheros que no tengan que sufrir modificaciones continuas.

En un fichero relativo (directo o aleatorio), el acceso a un registro individual no obliga a la gestión de los registros anteriores, por cuanto que se identifica directamente por el número del registro, es decir, por su posición en el fichero. Es, pues, posible la recuperación inmediata de los datos contenidos en un registro.

En un fichero con índice se elige un campo especial como clave de acceso del registro y, por ello, cada registro se identificará con el valor de su clave. Todos los índices están almacenados en un fichero separado, asociados a la dirección del registro al que hacen referencia. Cuando se requiere un registro se especifica su clave de acceso que, encontrada en el fichero separado, proporciona de forma inmediata los extremos para la identificación del registro deseado.

Con un fichero secuencial, la forma será:

```
SELECT          nombre_utilizado_en_el_programa
   ASSIGN[TO]   modo_selección , nombre_fiche-
                ro/nombre_dato
```


en donde modo_selección puede ser INPUT, OUTPUT, INPUT-OUTPUT, PRINT y RANDOM, según la operación a efectuar.

INPUT se utiliza para la lectura de los datos de un registro; OUTPUT, para la escritura en un registro, e INPUT-OUTPUT permite la realización simultánea de las dos operaciones. Para los ficheros de organización relativa y con índice se utilizará RANDOM, y para enviar datos a la impresora la modalidad preestablecida será PRINT.

Así, si se quisiera leer el fichero externo "TARJETAS" y hacer referencia al mismo mediante el nombre MI_FICHERO, se tendría:

SELECT MI-FICHERO ASSIGN TO INPUT,"TARJETAS"

mientras que para enviar datos a la impresora se podría tener:

SELECT IMPRESION-DATOS ASSIGN TO PRINT

las otras declaraciones relativas a los ficheros son:

ORGANIZATION
ACCESS
STATUS

ORGANIZATION tiene relación con el modo en el que se creó el fichero y no puede modificarse en el curso de las operaciones de proceso. Su sintaxis es:

ORGANIZATION [IS] SEQUENTIAL/RELATIVE INDEXED

dependiendo de las diferentes modalidades de creación de los ficheros. Las operaciones que se pueden efectuar en un fichero son la lectura y escritura de los datos contenidos. ACCESS indicará el orden en el que deben tratarse los registros. La sintaxis, en el caso secuencial, será:

ACCESS [MODE IS] SEQUENTIAL

en donde los registros se procesarán uno tras otro, mientras que, con ficheros relativos, la sintaxis será:

```
ACCESS [MODE IS]  SEQUENTIAL [RELATIVE KEY IS nom-
bre_dato]
                  RANDOM     [RELATIVE KEY IS nom-
bre_dato]
                  DYNAMIC
```

El acceso secuencial establece que los registros se procesarán uno tras otro, en el orden creciente de número de registro. Con el acceso aleatorio, el valor de RELATIVE KEY especificará cuál es el registro a tratar. Con el acceso dinámico el fichero podrá procesarse tanto en el modo secuencial como en el modo aleatorio.

Con ficheros con índice se tendrá:

```
ACCESS [MODE IS] SEQUENTIAL
                 RANDOM
                 DYNAMIC
RECORD [KEY IS]   nombre_dato
[ALTERNATE RECORD [KEY IS]
nombre_dato [WITH DUPLICATES]]
```

El acceso secuencial establece que los registros se procesarán uno tras otro, en orden de clave de acceso creciente. Con el acceso aleatorio el valor de RECORD KEY especificará cuál es el registro a tratar y con el acceso dinámico el fichero podrá procesarse tanto de manera secuencial como de manera aleatoria.

La declaración ALTERNATE RECORD especifica una clave de acceso alternativa para los registros del fichero. Si se especifica la opción WITH DUPLICATES, en el fichero podrán existir registros con la misma clave de acceso alternativa.

Es importante subrayar que cuando un fichero con índice se procesa el campo especificado como clave de acceso debe ser el efectivamente utilizado en el momento de la creación del fichero.

A falta de especificaciones expresas para un fichero se considerará que tanto la organización como el acceso son secuenciales.

Además, con el empleo de la opción

```
FILE STATUS [IS]    nombre_dato
```

después de cada operación efectuada en el fichero en la variable nombre_dato se encontrará un valor que dependerá del éxito de la operación. Así:

```
SELECT MI-FICHERO ASSIGN TO RANDOM, "TARJETAS"
ORGANIZATION IS RELATIVE
ACCESS MODE IS RANDOM, RELATIVE KEY IS CLAVE-ACCESO
    FILE STATUS IS INDICADOR-ERROR
```

informará al compilador de que se tiene la intención de procesar el fichero relativo "TARJETAS", haciendo referencia al mismo con

el nombre MI-FICHERO, de manera relativa, utilizando como clave el valor contenido en CLAVE-ACCESO y conservando en INDICADOR-ERROR un valor que dependerá del resultado, satisfactorio o no, de la operación efectuada.

De forma análoga:

```
SELECT ALMACEN ASSIGN TO RANDOM, "STOCK"
ORGANIZATION INDEXED
ACCESS DYNAMIC
RECORD KEY CODIGO
```

procesará el fichero con índice "STOCK" haciendo referencia al mismo con el nombre ALMACEN, con acceso tanto secuencial como aleatorio, utilizando como clave de acceso el campo denominado CODIGO.

El párrafo

```
I-O-CONTROL
```

indica las técnicas especiales a utilizar en la gestión de los ficheros. Las instrucciones que están incluidas en esta sección son, por ejemplo:

```
SAME [AREA FOR]   nombre_fichero_1, nombre_fichero_2,...
```

que establece que dos o más ficheros comparten la misma zona de memoria, aunque no sean del mismo tipo, o:

```
NO APPLY [AREA FOR]
nombre_fichero_1 , nombre_fichero_2 , ...
```

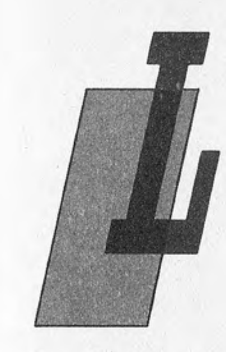
que ordena al compilador que no reserve zonas de memoria para los datos contenidos en los ficheros indicados, etc.

Se trata de una sección de utilización especial.

CAPITULO VIII

COBOL: LOS DATOS

DATA DIVISION (División de datos)



La división de datos describe la naturaleza y la estructuración de los datos utilizados en el programa, que pueden ser de tres tipos:

- datos contenidos en fichero;
- datos procesados por el programa y utilizados para conservar resultados intermedios o formatos de impresión;
- constantes definidas por el usuario.

Las tres secciones de las que está constituida la división tienen las funciones siguientes:

- **FILE SECTION** declara la estructura de los ficheros. Cada fichero se identifica por un bloque de descripción general o por uno de descripción del registro.
- **WORKING-STORAGE SECTION** describe registros y datos no estructurados que no forman parte de ficheros externos, pero que son procesados internamente por el programa, y constantes.
- **LINKAGE SECTION,** de la misma composición que la sección WORKING-STORAGE, tiene significado solamente cuando el procedimiento completo de los programas prevé que estos datos tengan que ser compartidos entre varias unidades de programa.

El conjunto de instrucciones relativas a la descripción de los ficheros tiene la sintaxis:

FD   nombre_del_fichero

Debe de haber una entrada de descripción de fichero (FD) por cada fichero que aparezca en una instrucción SELECT de la división del entorno, y sus nombres deben coincidir. Sus opciones son:

```
LABEL RECORD IS/RECORDS ARE    STANDARD/OMITTED
BLOCK [CONTAINS] num_1 [TO num_2] RECORDS/CHARACTERS
RECORD [CONTAINS] num_1[TO num_2] CHARACTERS
VALUE OF LABEL IS    nombre_dato
DATA RECORD IS/RECORDS ARE    nombre_1 [nombre_2] ...
```

La palabra FD debe escribirse en el área A y ha de ir seguida por el nombre del fichero utilizado por el programa. Las demás instrucciones comenzarán en la zona B. Hay un punto único, situado al final de la instrucción FD (por tanto, al final de todas sus opciones).

La instrucción BLOCK describe la dimensión de un registro físico, cuando no sea estándar, en términos de registros o, cuando la especificación de un número de registro sea imposible, en términos de caracteres. También está permitido indicar los límites de la dimensión de los bloques cuando la longitud no se tenga que considerar fija.

La instrucción RECORD suele omitirse por cuanto que el registro se describe ampliamente en las fases sucesivas.

La cláusula LABEL especifica la existencia, o no, de etiquetas para el fichero o el dispositivo al que está asignado el fichero. Pero los ficheros asignados a dispositivos RANDOM (aleatorios) la etiqueta ha de especificarse como STANDARD.

VALUE OF LABEL es un comentario que especifica el valor de la posible etiqueta del dispositivo o del fichero.

DATA RECORD especifica el nombre de las estructuras de datos que componen el fichero. Cada nombre asignado corresponderá al más alto nivel (01) posible en un fichero y deberá ser objeto de llamada en la siguiente sección de descripción de registro.

Es pues posible que un registro esté estructurado de modos diferentes, con la característica de que los datos de las diversas estructuras compartan la misma área.

En efecto:

DATA RECORDS ARE ESTRUC-1, ESTRUC-2, ESTRUC-3

indicará que la estructura del fichero puede examinarse según

tres modelos de subdivisión, a los que será posible hacer referencia con cualquier instrucción.

En las secciones sucesivas será necesario encontrar la descripción detallada de cómo están constituidas las tres estructuras.

Descripción del registro

A continuación de la descripción del fichero (FD) debe ir al menos una entrada de descripción del registro al nivel 01. Su propósito (tanto de este nivel como de los siguientes) es informar al compilador COBOL de la estructura del "registro lógico" utilizado. El número de nivel asignado a cada campo varía de 01 a 49, dando idea de pertenencia; el 01 se reserva para el nombre del registro y se empieza a escribir en el área A. Para facilitar la comprensión se suelen estructurar las líneas, sangrando los distintos subcampos pertenecientes a otro respecto de éste. Cuando un campo no tiene subcampos se denomina elemento simple, en contraposición a los que sí los poseen (elementos compuestos).

La descripción de un registro es igual en las tres secciones de DATA DIVISION y tiene la estructura siguiente:

número_nivel   nombre_dato-1/FILLER

seguido por las opciones:

```
REDEFINES       nombre_dato-2
PICTURE/PIC [IS] cadena_de_caracteres

USAGE [IS]      COMPUTATIONAL        /
                COMP                 /
                COMPUTATIONAL-1      /
                COMP-1               /
                COMPUTATIONAL-3      /
                COMP-3               /
                DISPLAY              /
                INDEX

SIGN IS TRAILING/LEADING [SEPARATE [CHARACTER]]

OCCURS  número_de_veces [TIMES]
        num_1  TO  num_2 [TIMES] DEPENDING [ON]
        nombre_dato [INDEXED [BY] nombre_índi-
        ce_1 , ...]
```

```
SYNCHRONIZED/SYNC  [LEFT/RIGHT]
JUSTIFIED/JUST     [RIGHT]
BLANK [WHEN] ZERO
VALUE [IS]    valor_dato
```

El número de nivel (que puede variar de 01 a 49) muestra el nivel jerárquico del dato en la estructura del registro. Además de ello diferencia un modelo de estructuración de un registro de una variable independiente, que tendrá siempre un número de nivel igual a 77.

Como se dijo anteriormente, si en un registro hay varios niveles 01 (el nivel máximo de "agregación") se deberán encontrar las correspondientes definiciones en el grupo de descripción del fichero encabezado por FD. Continuando con un ejemplo antes utilizado, se deberá encontrar:

```
FD  ...
    DATA RECORDS ARE ESTRUC-1, ESTRUC-2, ESTRUC-3
...
01  ESTRUC-1.
...
01  ESTRUC-2.
...
01  ESTRUC-3.
```

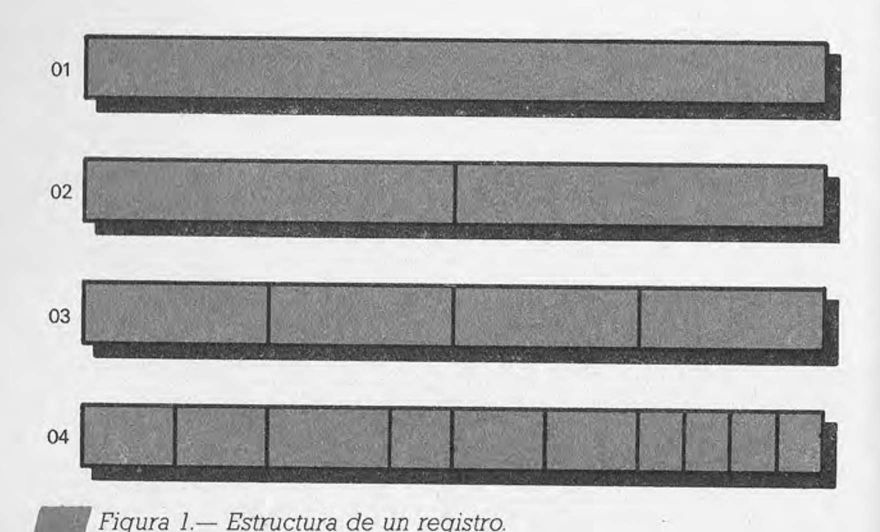


Figura 1.— Estructura de un registro.

Por consiguiente, en la estructura de la figura 1 serán atribuidos números de nivel crecientes a las estructuras que van descendiendo, poco a poco, a definiciones de mayor detalle hasta llegar a los datos elementales, para los cuales sólo se admiten instrucciones específicas sobre las características del contenido, tales como VALUE, PIC, etc. El nombre_dato que está junto al número de nivel tiene el significado de simbolizar la denominación del campo descrito, pero también representa todos los posteriores subniveles de "agregación". Por consiguiente, cada vez que en una instrucción aparezca el nombre de un campo de un registro, serán afectados todos los datos que estén bajo la denominación correspondiente. Es decir, después de haber asignado un nombre a un registro será posible, por ejemplo, imprimir todos los datos contenidos en dicho registro especificando su nombre como objeto de la instrucción de impresión.

Así:

```
FD  TARJETA
    LABEL RECORD IS OMITTED
    RECORD IS DATO

01  DATO.
  02  CAMPO_1                    PIC X.
  02  CAMPO_2
    03  SUB_CAMPO_1              PIC X.
    03  SUB_CAMPO_2              PIC X.
  03  CAMPO_3                    PIC X.
```

describirá el fichero TARJETA, que no tiene etiquetas (OMITTED significa "omitidas"), en el cual el único registro toma el nombre de DATO y está estructurado en tres campos, uno de los cuales está subdividido a su vez en otros dos. Una instrucción de lectura o de escritura en DATO implicará todos los datos elementales (CAMPO_1, SUB_CAMPO_1, SUB_CAMPO_2, CAMPO_3) que lo constituyen.

La palabra FILLER indica que el campo no tiene nombre y que no contendrá valores significativos. Por consiguiente, FILLER puede utilizarse para datos de nivel elemental, pero no para los de nivel máximo, como 01. En efecto, fuera del registro, incluso los datos de nivel 77 tienen la necesidad de estar definidos, por lo que para ellos no será posible el empleo de esta opción.

Redefines

La cláusula

nombre_dato_1 REDEFINES nombre_dato_2

especifica que un nombre_dato se refire a la misma zona de memoria ocupada por otro nombre_dato. Así:

```
01  A1.
 02  B1
  03  C1                          PIC X.
  03  C2                          PIC X.
 02  D1  REDEFINES B1
  03  E1                          PIC X.
  03  E2                          PIC X.
  03  E3                          PIC X.
 02  B2                           PIC X.
```

indica que, en memoria, se tendrá la estructura de la figura 2.

Los campos C1 y C2 ocupan la misma zona de E1, E2 y E3. Se podrá, pues, elegir hacer referencia a B1 o a D1, que lo redefine.

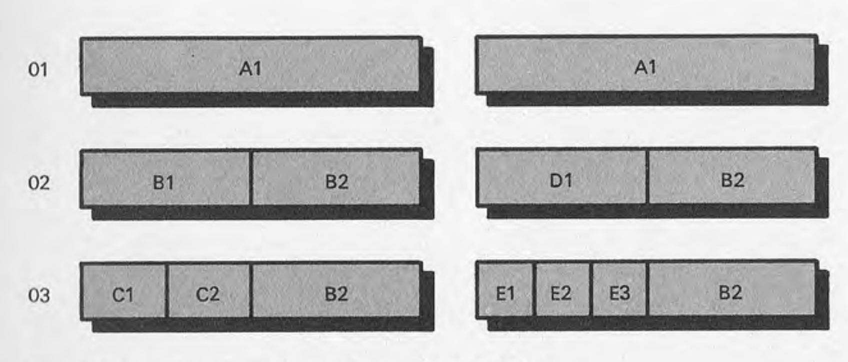


Figura 2.— Estructura al usar REDEFINES.

Picture

La cláusula PICTURE describe las características de representación de un dato elemental (elemento simple) y tiene como objeto un conjunto de caracteres (hasta de 30), que "direccionan" al

compilador para comprender el número y la calidad de los elementos que lo distinguen.

Las categorías de datos que pueden describirse por PICTURE (o PIC) son las siguientes:

- Alfabético.
- Numérico.
- Alfanumérico.
- Alfanumérico editado.
- Numérico editado.

Para un dato alfabético, la cadena característica de PIC sólo puede contener "A" y "B"; el contenido del dato alfabético será cualquier combinación de las letras del alfabeto inglés y espacios.

El número de caracteres que el dato contiene viene indicado entre paréntesis:

```
04 EJEMPLO_1      PIC A(8).          (o PIC AAAAAAAA)
```

se utilizará para indicar que EJEMPLO_1 es un campo alfabético de 8 caracteres.

Para un dato numérico, la cadena característica tendrá un "9" por cada cifra (hasta un máximo de 18) y símbolos especiales, tales como "S", que indica la presencia de un signo; "V", que indica la posición del punto decimal, y "P", que indica la posición a la que debe desplazarse la coma (PPP indicará un desplazamiento de tres lugares decimales). Así:

```
04 EJEMPLO_1      PIC 9(7)           (o PIC 9999999)
```

indicará un número de 7 cifras, mientras que

```
04 EJEMPLO_1      PIC S99V999.
```

indicará un número de 5 cifras, de las cuales la primera comprende el signo y las tres últimas representan la parte fraccionaria.

Por último:

```
04 EJEMPLO_1      PIC 99PPP.
```

indicará que el número de 2 cifras efectivo se representará multiplicado por 1000 (con desplazamiento de tres lugares decimales).

El contenido de un dato numérico será, pues, una combinación de las 10 cifras y, si está presente S, de los símbolos + y –.

Para un dato alfanumérico, que podrá contener cualquier com-

binación de los caracteres del juego del ordenador, la cadena característica suele estar constituida por "X" (ocasionalmente en unión con "A" y "9") y por el número de caracteres que representa su dimensión.

Para un dato alfanumérico editado la cadena representativa debe contener al menos:

una X y una B
una X y un 0
una X y un /
una A y un 0
una A y un /

y también valores "9"; su contenido será una combinación de los caracteres del juego del ordenador.

Para un dato numérico editado, la cadena característica puede contener solamente 0, 9, B, P, V, Z, /, *, +, -, CR, DB, $, el punto y la coma. En efecto, tendrá que contener al menos uno de los símbolos 0, B, /, Z, *, +, -, CR, DB, $, el punto y la coma, y el contenido del dato estará constituido por números.

Los datos editados sólo se utilizan en operaciones de transferencia e impresión de datos en PROCEDURE DIVISION.

El significado de cada símbolo dentro de la cadena característica de PIC es:

- A representa una posición en la que insertar una letra del alfabeto o un espacio.
- B representa una posición en la que introducir un espacio.
- P indica un factor de escala 10 por el que multiplicar (si está a la derecha) o dividir (si está a la izquierda) el número dado.
- S indica la presencia del signo.
- V indica la presencia y la posición del punto decimal, pero no ocupa una posición adicional en la dimensión del dato.
- X indica una posición en la que insertar cualquier carácter.
- Z indica una posición numérica que, cuando contiene un 0 no significativo, se representa por un espacio (ocupa una posición efectiva).
- 9 indica una posición en la que introducir una cifra.
- 0 indica una posición en la que insertar un 0.

- / indica una posición en la que introducir una barra (/).
- , indica una posición en la que introducir una ",".
- . indica una posición en la que insertar un "." que no sea la posición del punto decimal. A diferencia con V, la posición se contabiliza en la longitud de la cadena. Cuando se especifica la opción DECIMAL POINT IS COMMA, en SPECIAL-NAMES, las atribuciones del punto y de la coma están completamente invertidas.
- + indica una posición en la que introducir un espacio si el número es positivio ó 0, o, si el número es negativo, un signo "–".
- – indica una posición en la que introducir un signo "+" si el número es positivo ó 0, o, si el número es negativo, un signo "–".
- CR indica la posición en la que introducir dos espacios si el número es positivo ó 0, y CR si el número es negativo.
- DB indica la posición en la que insertar dos espacios si el número es positivo ó 0, y DB si el número es negativo.
- * indica una posición numérica que, cuando contiene un cero no significativo, se representa por un asterisco (ocupa una posición efectiva).
- $ (o el símbolo correspondiente establecido en CURRENCY SIGN IS en SPECIAL-NAMES) indica la posición en la que introducir el símbolo de la moneda.

El símbolo "$" debe ser el primer carácter de la cadena (si no existen también "+" o "–"), mientras que "+" y "–" pueden ser el pri-

ORIGEN		DESTINO	
Dato inicial	PICTURE	PICTURE	Dato final
45378	S9(5)	–ZZ,ZZ9.99	45,378.00
78	S99999V	–ZZ,ZZ9.99	78.00
0	9(5)	$ZZ9.99	.00
0	9(5)	$ZZZ.ZZ	
5378	9(5)	$**,**9.99	$*5,378.00
378	9(5)	$**,**9.99	$***378.00
378	999V99	$$,$$9.99	$3.78
–5378	S9(5)	–ZZ,ZZ9.99	–5.378.00
–5378	S9(5)	ZZZZZ.99CR	5378.99CR
45378	S9(5)	99BB999	45 378
0	9(5)	$*,***.***********	

Figura 3.— Datos numéricos editados.

mer y el último carácter y CR y DB, finalmente, deben aparecer en la última posición de la cadena.

El signo V y el punto decimal se excluyen recíprocamente, así como +, -, CR, DB y Z, *.

Cuando los signos de edición están repetidos, indican que el signo debe preceder a la primera cifra efectiva.

Así, considerando la representación inicial y final de un dato numérico editado, se tendrán como ejemplos los mostrados en la figura 3.

Usage

La cláusula USAGE IS tiene relación con el formato de representación interna de los datos. Si la cláusula especificada es:

USAGE IS DISPLAY

puesto que el dato está destinado a visualizarse en la pantalla se conservará con el empleo de una unidad de memoria por cada carácter individual. Por consiguiente, un valor numérico se conservará exactamente como se muestra al exterior, es decir, un carácter por cada cifra.

Cuando se utilizan muchos valores numéricos, puede ser de utilidad para ahorrar espacio en memoria emplear su representación binaria, conservándoles de forma "empaquetada". Se utilizarán entonces las diversas formas COMPUTATIONAL, con las cuales cada cifra se conserva en la mitad de una unidad de memoria, con el resultado de reducir a la mitad la ocupación correspondiente.

Una utilización especial es:

USAGE IS INDEX

por cuanto que las variables indicadas como INDEX no están sujetas a las reglas que afectan a las variables normales y, por ello, no podrán inicializarse con las instrucciones normales.

Es importante recordar que la cláusula USAGE, aplicada a un campo de un registro, actúa sobre todos los datos de los niveles inferiores al mismo.

Las demás cláusulas

La cláusula SIGN indica la posición y la representación del signo cuando está especificado en la cláusula PICTURE del dato

al que se refiere y cuando por ello se haya declarado USAGE IS DISPLAY. Puede seguir (TRAILING/LEADING) al valor numérico o ser un carácter separado (TRAILING/LEADING SEPARATE).

La cláusula OCCURS especifica un conjunto de campos del mismo tipo a los cuales se hace referencia mediante un índice.

La sintaxis:

OCCURS número_de_veces [TIMES]
 [INDEXED [BY] nombre_índice_1 , nombre_índice_2 , ...]

o bien:

OCCURS num_1 TO num_2 [TIMES] DEPENDING [ON]
nombre_dato [INDEXED [BY] nombre_índice_1 , ...]

indica que un campo es un vector de datos homogéneos o una tabla y que el elemento del vector está identificado por el valor de nombre_índice_1, nombre_índice_2, etc. El número de elementos del vector está establecido por número_de_veces o bien puede variar entre num_1 y num_2, dependiendo del valor tomado por nombre_dato.

Así:

04 ELEMENTO_F1 OCCURS 4 TIMES INDEXED BY IX1.

representará lo mostrado en la figura 4.

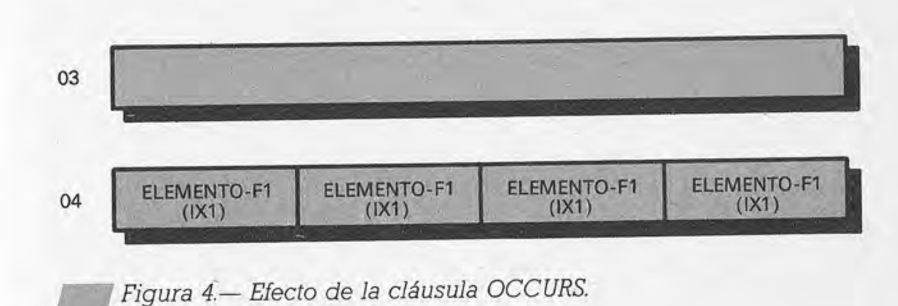


Figura 4.— Efecto de la cláusula OCCURS.

En donde el valor de IX1 especificará a qué elemento referirse.

La cláusula SYNCHRONIZED se refiere a la ocupación de la celda de memoria por parte de un dato. LEFT indicará que el dato ocupará la parte de memoria que le compete a partir de la izquierda y, si queda todavía espacio disponible, se introducirá el

número oportuno de espacios para impedir que otros datos se coloquen en la parte de memoria sobrante. RIGHT indicará que el mecanismo de colocación del dato, aun permaneciendo el descrito, tomará el desplazamiento de la derecha. A falta de indicaciones, será válida la opción LEFT.

La cláusula JUSTIFIED, válida solamente para datos alfabéticos o alfanuméricos no editados, indica que si un dato es más extenso que la variable que lo tiene que acoger, se perderán los caracteres sobrantes a la izquierda. Esta opción solamente será válida cuando se quieran modificar los criterios normales de truncamiento (a la derecha) y de alineación (a la izquierda) estándar de los datos elementales.

La cláusula BLANK [WHEN] ZERO especifica que un dato, si su valor es cero, estará representado por espacios. Se trata de una cláusula que no puede "convivir" con una cláusula PICTURE en la que existan asteriscos. Se aplica solamente a datos elementales numéricos.

La cláusula VALUE establece el valor inicial de un dato. Por supuesto, el valor proporcionado deberá ser compatible con la representación de PICTURE establecida.

Datos independientes

Todos los datos que no pertenecen a un registro se distinguen por tener un número de nivel igual a 77.

Para dichos datos son válidas todas las opciones establecidas para los datos elementales, es decir:

PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, BLANK, VALUE

Veamos un ejemplo de una declaración completa:

```
IDENTIFICATION DIVISION.
PROGRAM-ID DATA_1.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER...
OBJECT-COMPUTER...
SPECIAL-NAMES.
          DECIMAL POINT IS COMMA.

INPUT-OUTPUT SECTION.
```



```
FILE CONTROL.
     SELECT TARJETA-ENTRADA
        ASSIGN TO RANDOM, "TARJETAS"
        ORGANIZATION IS INDEXED
        ACCESS IS DYNAMIC
        RECORD KEY IS NOME

     SELECT HOJA
        ASSIGN TO PRINT


DATA DIVISION.

FILE SECTION.
FD TARJETA-ENTRADA
        BLOCK CONTAINS 2 RECORDS
        RECORD CONTAINS 128 CHARACTERS
        LABEL RECORDS ARE STANDARD
        DATA RECORD IS TARJETA

FD HOJA
        VALUE OF MARGEN-SUP                    IS 6
        VALUE OF MARGEN-INF                    IS 6
        VALUE OF LINEAS-PAGINA                 IS 66
        VALUE OF INTERLINEA                    IS 1
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS LINEA


01 TARJETA.
   02 DATOS-ANAGRAFICOS.
      03 NOMBRE                         PIC X(30).
      03 DIRECCION                      PIC X(30).
      03 EDAD                           PIC 9(3).
      03 TELEFONO OCCURS 4 TIMES.       PIC X(10).
         04 DOMICILIO                   PIC 9(4).
         04 PREFIJO                     PIC 9(8).
         04 NUMERO
      03 CIUDAD.                        PIC 9(6).
         04 CODIGO POSTAL               PIC X(24).
         04 MUNICIPIO
   02 DATOS-CONTABLES.
      03 TOTAL-MES OCCURS 12 TIMES INDEXED BY MES.
         04 SALDO                       PIC-ZZ.ZZZ.ZZZ.ZZ9
```

```
        03 TOTAL-ANO                    COMP-3 VALUE IS 0.
                                        PIC-ZZ.ZZZ.ZZZ.ZZ9
                                        COMP-3 VALUE IS 0.
    02 FILLER                           PIC X(99).
01 LINEA                                PIC X(132) VALUE IS
                                        SPACE.

WORKING-STORAGE SECTION.
77 I1                                   PIC 9(2) VALUE IS 1.
77 J1                                   PIC 9(2) VALUE IS 0.
77 SENAL                                PIC X VALUE IS "N".
77 MES                                  PIC 99 VALUE IS 1.
77 NOMBRE                               PIC X(30).
77 I                                    INDICE.
78 J                                    INDICE.

01 LINEA-IMPRESION.
   02 NUM-ORDEN
   02 ANAGRAFICA                           PIC 9(4).
      03 NUMERICA                          PIC X(100).
                                           PIC X(28).

01 TABLA.
   02 LINEA-MATRIZ OCCURS 12 TIMES.
      03 COLUMNA-MATRIZ OCCURS 3 TIMES PIC-ZZ.ZZZ.ZZZ.ZZ9.
```

En este ejemplo están simbolizadas las opciones de más frecuente utilización.

CAPITULO IX

COBOL: INSTRUCCIONES

Procedure Division (División de procedimientos)

Una vez definido el programa, especificado el ambiente exterior al mismo y las características de Entrada/Salida y definida, así como la naturaleza y estructura de todos los datos que se utilizarán, al compilador no le queda sino conocer el modo en el que los datos se procesarán y modificarán o, lo que es lo mismo, las instrucciones efectivas de ejecución.

PROCEDURE DIVISION puede dividirse en secciones, párrafos y segmentos (o frases), terminando normalmente con la instrucción END PROGRAM.

Las secciones y los párrafos son encabezamientos de grupos de instrucciones que es posible activar haciendo referencia a su denominación, y sus cabeceras se escriben en la zona A.

Así:

```
PROCEDURE DIVISION.
...
...
MUESTRA SECTION
     instr._1
     instr._2
     ...
     instr._N
...
END PROGRAM.
```

indica que si en una instrucción existe un reenvío a MUESTRA se ejecutarán las instrucciones 1, 2, ..., N contenidas en ella.

Los segmentos, numerados de 0 a 127, se utilizan para especificar qué parte del programa quedará siempre en memoria, por cuanto que se define de forma implícita como área fija (los segmentos 0 a 49) y qué parte se cargará solamente cuando se solicite por instrucciones específicas de referencia (50 en adelante).

Se admite también la inserción de una sección DECLARATIVES, en la cual se incluirá la instrucción USE que se empleará para indicar, en caso de error, qué subrutina deberá ejecutarse, definida en el ámbito de la misma sección, cuando no esté prevista con anterioridad entre las cláusulas de los comandos un procedimiento de tratamiento del error. Su sintaxis será:

```
PROCEDURE DIVISION.
DECLARATIVES.
CONTROL-I-O SECTION.
  USE AFTER ERROR PROCEDURE I-O.
  GESTION-ERROR.
  DISPLAY "ERROR EN EL PROCEDIMIENTO".
  STOP RUN.
END DECLARATIVES.
END PROGRAM.
```

La sección DECLARATIVES debe terminar con END DECLARATIVES.

La instrucción más utilizada en COBOL es, seguramente,

```
MOVE valor TO var_1 , var_2,...
```

que realiza la asignación de un valor o un dato. Así es posible la inicialización de varios datos, alfabéticos y numéricos, en cualquier punto del programa.

La única restricción se tiene con las variables en las que USAGE esté definido como INDEX, en cuyo caso la instrucción correspondiente ha de ser SET.

Así:

```
MOVE "TITULO" TO ENCABEZAMIENTO
MOVE SPACE TO CABECERA
MOVE ZERO TO CONTADOR
MOVE 0 TO A,B,C,D(7)
MOVE 45.7 TO NUMERO
```

son válidas si ENCABEZAMIENTO y CABECERA están definidas como variables alfabéticas y CONTADOR,A,B,C, el vector D( ) y NUMERO, lo están como variables numéricas.

Es importante transferir también un valor a una variable no

elemental que represente a un conjunto de datos. No obstante, para las transferencias complejas es preferible utilizar la opción

```
MOVE CORRESPONDING nombre_1 TO nombre_2
```

con la cual los datos se transferirán con la debida atención a la composición de los grupos.

Operaciones aritméticas

Las operaciones aritméticas son:

- AND
- SUBTRACT
- MULTIPLY
- DIVIDE

que tienen una sintaxis que respeta perfectamente las expresiones inglesas correspondientes.

Para ellas existen dos formas alternativas según que se tenga que introducir o no el resultado de la operación en una nueva variable. Así:

```
ADD 1 TO PARCIAL
```

introducirá en la variable PARCIAL el valor anterior incrementado en 1, mientras que

```
AND PARCIAL-1,PARCIAL-2 GIVING TOTAL
```

efectuará la suma de PARCIAL-1 y PARCIAL-2, introduciendo el resultado en TOTAL.

Una opción adicional común a todas las operaciones aritméticas es la proporcionada por

```
ROUNDED
```

que especifica el redondeo a realizar para un número en correspondencia con la última cifra indicada en su PICTURE. Si no fuera así, se provocaría el truncamiento, en el caso de que el número de cifras resultante fuera superior al especificado.

También es posible indicar

```
ON SIZE ERROR Instrucción_a_ejecutar
```

que especifica una instrucción a ejecutar cuando se verifique un error de superación de la dimensión máxima prevista para un valor numérico. Cuando exista esta opción, el valor numérico "erróneo" se mantendrá para poder examinarlo, en lugar de hacerse indefinido.

Así, en el caso

```
ADD SUMANDO-1,SUMANDO-2,SUMANDO-3 GIVING TOTAL
   ROUNDED ON SIZE ERROR STOP RUN
SUBTRACT SUSTRAENDO FROM MINUENDO GIVING
DIFERENCIA
   ROUNDED
MULTIPLY FACTOR-1 BY FACTOR-2 GIVING PRODUCTO
DIVIDE DIVIDENDO BY DIVISOR GIVING COCIENTE
   ROUNDED ON SIZE ERROR DISPLAY"ERROR"
```

las variables TOTAL, DIFERENCIA, PRODUCTO y COCIENTE contendrán los resultados de las operaciones respectivas.

La instrucción COMPUTE calcula expresiones algebraicas complejas, en las cuales existen los operadores:

+	suma
-	resta
*	multiplicación
/	división
**	potencia
( )	paréntesis

que respetan los niveles de prioridad establecidos por el álgebra (y respetados, asimismo, por el lenguaje Fortran), es decir:

paréntesis
potencias
multiplicaciones y divisiones
sumas y restas

Así:

```
COMPUTE   RESULTADO = 3 + 5 * 6 - (DATO + INCREMENTO)
```

con DATO = 5 e INCREMENTO = 2, proporcionará en la variable RESULTADO el valor 26 (es decir, 3 + 30 - 7).

Para todas las operaciones algebraicas la representación de los números vendrá determinada por el tipo de operandos especificado en USAGE IS de las respectivas descripciones, teniendo, pues, valores COMP si se especifican expresamente en las definiciones de todos los datos.

La instrucción IF

También en el lenguaje COBOL existe la posibilidad de examinar el valor de un dato, el resultado de una expresión aritmética o el sentido de una comparación entre dos elementos.

La sintaxis general es:

```
IF condición   NEXT SENTENCE/instrucción_1
      ELSE     NEXT SENTENCE/instrucción_2
```

en donde, si la condición indicada es verdadera, se ejecuta la instrucción sucesiva (NEXT SENTENCE) o la indicada por la instrucción_1 y, por el contrario, si la condición no es verdadera, se ejecutará cuanto sea especificado a continuación de ELSE, es decir, la instrucción sucesiva o la indicada por la instrucción_2.

Las condiciones pueden expresarse por una variable, una expresión aritmética o por una expresión condicional.

Una expresión condicional puede tomar solamente dos valores, verdadero o falso, según que cuanto sea indicado por la expresión se verifique por las relaciones existentes entre los valores de las magnitudes implicadas. La forma general de una expresión condicional simple puede ser:

```
elemento_1   IS EQUAL / =        TO    elemento_2
             IS NOTEQUAL         TO
             IS GREATER / >      THAN
             IS NOT GREATER      THAN
             IS LESS / <         THAN
             IS NOTLESS          THAN
```

con el significado siguiente:

A IS EQUAL TO B	proporcionará un valor verdadero si A = B.
A IS NOT EQUAL TO B	verdadera si A es diferente de B.
A IS GREATER THAN B	verdadera si A es mayor que B.
A IS NOT GREATER THAN B	es verdadera si A es menor o igual a B.
A IS LESS THAN B	es verdadera si A es menor que B.
A IS NOT LESS THAN B	verdadera si A es mayor o igual a B.

Varias expresiones condicionales simples pueden unirse mediante los operadores AND,OR y NOT que especifican, respecti-

vamente, que la expresión compleja resultante será verdadera si todas las expresiones simples que la constituyen son verdaderas (AND), si al menos una de las expresiones simples se verifica (OR) o si es verdadero su inverso (NOT).

Así:

A IS EQUAL TO B AND C IS NOT EQUAL TO D

será verdadera si A = B y C diferente de D, mientras que:

A IS EQUAL TO B OR M(1) IS GREATER THAN 0 OR H IS NOT EQUAL TO Q

será verdadera si A = B, si M(1)>0 o si H = Q, mientras que:

NOT A = B

será verdadera si es falso que A es igual a B.

Otra forma de las expresiones condicionales se refiere a la naturaleza de un dato, que puede ser "indagada" para valorar si se trata de un dato numérico o alfabético con:

dato IS / IS NOT NUMERIC/ALFABETIC

mientras que para los datos solamente numéricos es posible averiguar si:

```
dato "IS / IS NOT   POSITIVE
                    NEGATIVE
                    ZERO
```

es decir, si el dato es mayor, menor o igual a 0.

Cuando se tiene una expresión condicional compleja, el orden de resolución de la expresión sigue el criterio de prioridad asociado a los operadores condicionales, es decir:

paréntesis
expresiones aritméticas
valoración expresiones condicionales simples
valoración AND y OR.

En definitiva, el procedimiento

```
PROCEDURE DIVISION.
ASIGNACIONES-INICIALES
    MOVE ZERO TO A , B
```



```
    MOVE 4 TO M(1)
    MOVE "INICIO" TO HACHE
    MOVE "FIN" TO Q.
TEST.
    IF A=B AND M(1)>0 AND HACHE NOT EQUAL TO Q
        DISPLAY "TODO CORRECTO".
    ELSE DISPLAY "RESULTADO NEGATIVO".
END PROGRAM.
```

producirá la impresión de la frase TODO CORRECTO, puesto que el test examina una condición que resulta ser verdadera.

De cualquier modo, una vez terminada la sección en la que se encuentra IF, se ejecutará la instrucción inmediatamente posterior.

PERFORM

Por lo que respecta a la ejecución de bucles repetitivos bajo el control de una variable, la sintaxis es:

PERFORM proc_1 [THROUGH/THRU proc_2]

con las opciones

VARYING var_1 FROM val_1
BY incr_1 UNTIL condic_1

que especifica al compilador que el programa ejecutará todas las instrucciones contenidas entre el encabezamiento del procedimiento proc_1 y el de proc_2, suponiendo que var_1 es la variable de control, cuyo valor inicial será val_1, el bucle se ejecutará el número de veces que resulta de incrementar var_1 en incr_1 al final de cada bucle hasta que sea falsa la condición condic_1. Tan pronto como esta condición sea verdadera terminará el bucle.

Así:

```
PROCEDURE DIVISION
INICIO
    PERFORM BUCLE THRU FIN-BUCLE
    VARYING I FROM 0 BY 2
    UNTIL I IS GREATER THAN 10
BUCLE.
    DISPLAY "REPETICION NUMERO".
    DISPLAY (I/2) + 1
```

```
FIN-BUCLE.
    EXIT.
END PROGRAM.
```

producirá, en el curso de 6 bucles, la impresión de los números desde 1 a 6, por cuanto que las instrucciones desde BUCLE a FIN-BUCLE se ejecutan evaluando el valor que toma I que, partiendo de 0, se incrementa en 2 al final de cada bucle y el bucle se repetirá mientrs I sea menor o igual a 10.

Otra forma de PERFORM indica directamente cuántas veces debe ejecutarse el bucle; así:

```
PERFORM BUCLE THRU FIN-BUCLE 2 TIMES
```

haría que se ejecutara BUCLE dos veces, sin ninguna necesidad de examinar variables de control.

Es posible que un bucle PERFORM contenga, a su vez, otro bucle, pero es importante que un bucle PERFORM no tenga una instrucción final fuera del bucle PERFORM que le es más externo. Cuando el bucle prevea la modificación de varios índices se podrá utilizar la forma:

```
PERFORM proc_1 THRU proc_2
    VARYING var_1 FROM val_1
    BY incr_1 UNTIL condic_1
    AFTER var_2 FROM val_2
    BY incr_2 UTIL condic_2
    ...
```

en donde el bucle se realiza también bajo control de var_2.

En realidad el bucle se ejecuta exactamente como antes, pero si condic_1 es falsa, el bucle proseguirá examinando la condición condic_2 y el bucle se realizará bajo el control del valor var_2. Tan pronto como sea verdadera condic_2 el bucle volverá a examinar condic_1, y si la encontrase falsa, recomenzará desde el principio, mientras que si la encontrase verificada terminará de forma definitiva.

Así, se tendrá el diagrama de bloques de la figura 5.

En condiciones normales se acopla a PERFORM la instrucción EXIT, que determina el punto común de término para varios procedimientos. Por consiguiente, se podrá encontrar con frecuencia en un programa

```
FINE-PROC.
    EXIT.
```

como instrucción terminal de referencia de un bucle PERFORM.

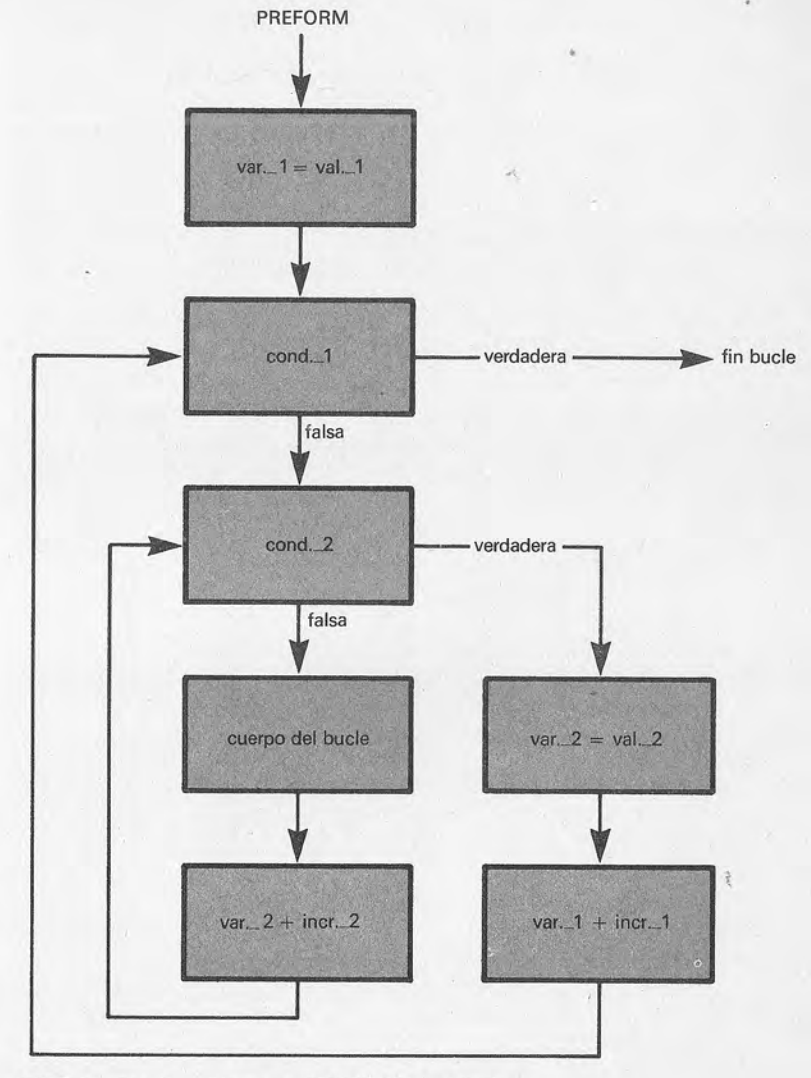


Figura 5.— Bloques PERFORM anidados.

Display

La instrucción que permite al programa mostrar en la pantalla informaciones, textos y datos es DISPLAY.

Su sintaxis permite definir la posición, la forma y el modo en que se presentarán los datos en la pantalla.

En la sintaxis general:

```
DISPLAY   dato  [UNIT  nombre_unidad]
  [LINE   número_línea POSITION número_columna
   SIZE   número_caracteres
   BEEP   ERASE
   HIGH/LOW BLINK REVERSE]
```

el argumento "dato" contiene el dato a presentar en pantalla y UNIT, si se indica, el terminal en el que deberá visualizarse el dato. En algunas versiones de COBOL se puede utilizar, con tal objeto, la cláusula UPON TERMINAL.

Es oportuno destacar que para que un dato sea susceptible de presentación en pantalla debe declararse que USAGE IS DISPLAY.

LINE indica la línea y POSITION indica la columna en correspondencia de las cuales el dato deberá visualizarse. A falta de estas indicaciones, el dato se imprimirá en la línea inmediatamente posterior a la posición actual del cursor o, si POSITION = 0, exactamente en donde se encuentra en ese momento.

SIZE especifica el número de caracteres que la presentación visual asociará a "dato" y que serán objeto de impresión. Si es menor que el contenido efectivo de dato, serán visualizados los caracteres que quepan a partir de la izquierda. SIZE, si se indica, tiene un "poder" superior a la indicación dimensional de PICTURE en el caso de discrepancia entre sus valores.

BEEP especifica la ejecución de una señal acústica y ERASE el borrado de toda la pantalla antes de la impresión de dato.

HIGH especifica que el dato (si el terminal permite estas definiciones) será presentado con doble luminosidad, LOW que será mostrado con una intensidad luminosa normal, BLINK que el rótulo será parpadeante y REVERSE que los caracteres serán, en vez de luminosos sobre el fondo oscuro como es habitual, oscuros sobre campo luminoso.

Así:

```
DISPLAY "Este es un rótulo"
    LINE 10 POSITION CRS-POSIT
    BEEP
    HIGH BLINK
```


producirá la frase: "Este es un rótulo" en la línea 10 y en la columna indicada por el valor de CRS-POSIT, generando una señal acústica antes de la impresión y mostrando el rótulo parpadeante en doble luminosidad.

ACCEPT

La instrucción ACCEPT permite adquirir directamente desde el terminal el valor a introducir en una variable.

Su sintaxis es:

```
ACCEPT dato [UNIT nombre_unidad]
   [LINE número_línea POSITION número_columna
   SIZE número_caracteres PROMPT símbolo
   ECHO CONVERT TAB ERASE NO BEEP OFF
   HIGH/LOW BLINK REVERSE
   ON EXCEPTION nombre_1 instrucción_1]
```

Para que sea válida, USAGE del dato no debe ser INDEX. Como en el caso de DISPLAY, a falta de UNIT que especifique el puesto de trabajo desde cuyo lugar se obtendrá la adquisición se sobrentiende que la operación tendrá lugar en correspondencia con el terminal que ha "lanzado" el programa.

LINE y POSITION especifican la posición en la pantalla en la cual se realizará la aceptación del dato, con las mismas peculiaridades de DISPLAY, es decir: indicar, a falta de especificaciones, la línea sucesiva o la misma posición (POSITION 0) en la que se encuentra el cursor.

SIZE, si se especifica, indica el número de caracteres que serán aceptados por el terminal y tiene una prioridad superior a la de PICTURE del dato elemental. Las reglas de aceptación siguen las de JUSTIFICATION especificadas o las de MOVE, es decir, el dato numérico alineado a la derecha y el dato alfabético alineado a la izquierda.

PROMPT especifica que la entrada del dato será guiada por la existencia de un número de símbolos igual a SIZE. Cuando no se especifica el carácter de PROMPT, será el guión de subrayado quien rellene el campo a partir del cual se extraerá el dato. La opción ECHO especifica que el contenido efectivo del dato, después de las alineaciones y conversiones de costumbre, se visualizará en la posición indicada.

CONVERT, para los valores numéricos, realiza la conversión en un número con el signo en la posición de la derecha y con la escala adecuada especificada.

TAB hace que la entrada no termine cuando se alcanza el final del campo especificado, sino que sea necesario un RETURN (o ENTER) para especificar que el dato se adquirió de modo definitivo. También será posible aceptar un BACKSPACE (RETROCESO), que reposiciona la entrada al carácter inmediatamente anterior y un TAB que lleva la entrada al primer carácter del campo.

ERASE indica que la pantalla se borrará antes del posicionamiento del campo de entrada.

NO BEEP suprime la señal acústica que se suele emitir en correspondencia con una instrucción de ACCEPT.

OFF anula la presentación visual del contenido del dato adquirido desde el teclado (se suele emplear para las palabras de acceso a datos protegidos contra consultas no autorizadas).

HIGH, LOW, BLINK y REVERSE muestran, como para DISPLAY, el contenido en luminosidad intensa, normal, parpadeante o en campo inverso.

ON EXCEPTION actúa si un carácter irregular se introduce desde el teclado. En consecuencia, el valor irregular se inserta en la variable nombre_1 y esta última puede examinarse por instrucciones sucesivas para valorar la naturaleza de la irregularidad.

Así:

```
ACCEPT  VALOR-DE-PARTIDA
    LINE  LIN  POSITION COL
    SIZE 10    PROMPT
    ECHO
    CONVERT
    TAB
    REVERSE  BLINK
    ON EXCEPTION CONTROL EXAMEN-ERROR
```

hará que sea posible adquirir de la pantalla el valor a introducir en la variable VALOR-DE-PARTIDA. El campo de entrada se posicionará en la columna y en la fila especificadas por las variables COL y LIN, se podrán tomar 10 caracteres como máximo o todos los que precedan al RETURN. Antes de la introducción se visualizarán 10 trazos de subrayado en la posición a partir de la cual adquirirá el dato, y una vez efectuada la conversión numérica se mostrará, en el mismo punto, el valor efectivo de VALOR-DE-PARTIDA, en campo inverso y con los caracteres parpadeantes. Será posible teclear también retrocesos (BACKSPACE) y tabulaciones (TAB) para corregir los valores tecleados. Si se escribiera un carácter irregular será englobado en CONTROL y la ejecución del programa se restablecerá por el procedimiento EXAMEN-ERROR.

GOTO y STOP

La instrucción GOTO realiza la transferencia del control del programa al procedimiento especificado, y así:

```
CALCULOS.
    ADD A TO B.
    MULTIPLY B BY 3 ROUNDED.
    GOTO FIN-PROGRAMA.
TODAVIA-MAS-CALCULOS.
    DIVIDE 4 BY 1.5
    ROUNDED.
FIN-PROGRAMA.
    EXIT.
END PROGRAM.
```

no ejecutará la sección TODAVIA-MAS-CALCULOS por cuanto que GOTO obligará al programa a ejecutar FIN-PROGRAMA.

También es posible una transferencia condicionada al valor de un dato, lo que se realiza con:

```
GOTO proc_1 , proc_2 , ...
     DEPENDING ON val_1
```

en donde si val_1 tiene el valor 1, el control se transferirá a proc_1, si tiene el valor 2 se transferirá a proc_2, y así sucesivamente. Si val_1 no tiene un valor utilizable, el control pasará a la instrucción sucesiva.

Para interrumpir la ejecución de un programa, la instrucción a especificar es:

```
STOP
```

que puede ir seguida por la palabra RUN o por una frase que se imprimirá en la pantalla.

SET

En caso de manejo de tablas y dado que un elemento de una tabla puede puede identificarse por su número de orden o por medio de índices, es necesario destacar el hecho de que un índice, en lenguaje COBOL, se trata de manera particular, por cuanto que al mismo se asocia un valor de 2 bytes, que corresponde a PIC 9(4) COMP cuando USAGE IS INDEX.

Por consiguiente, para un índice no pueden indicarse las opciones características de los datos elementales, tales como VALUE, PIC, etc., y, por ello, su valor será indefinido en el momento de la declaración.

No obstante, para poder atribuir un significado y, por consiguiente, un valor a los vínculos entre índices y elementos de una tabla, se utiliza la instrucción:

```
SET nombre_índice_1, nombre_índice_2,... TO nombre_1
```

o bien:

```
SET nombre_índice_1 , ... UP BY/DOWN BY nombre_1
```

que establecen que nombre_índice_1, etc., debe tener el mismo valor al que se refiere nombre_1 o que ha de incrementarse o decrementarse en el valor al que se refiere nombre_1.

CALL

En COBOL es posible que un programa llame a otro y que le transfiera, sin modificar, el valor de algunos datos. Para conseguirlo se hace referencia al nombre declarado en PROGRAM-ID, a algunas opciones adicionales especificadas en PROCEDURE DIVISION y a la introducción de la lista de los datos compartidos en LINKAGE-SECTION de DATA DIVISION del programa objeto de llamada.

Para conseguir, por ejemplo, que el programa "A" llame al programa "B" y le transfiera el valor de las variables COM-1 y COM-2, deberá especificarse en PROCEDURE DIVISION:

```
CALL "B" USING COM-1, COM-2
```

y en el programa B:

```
PROGRAM-ID.B.
...
...
DATA DIVISION.
...
...
LINKAGE-SECTION.
77 COM-1          PIC ...
77 COM-2          PIC ...
```



```
PROCEDURE DIVISION USING COM-1, COM-2.
...
...
EXIT PROGRAM.
```

por cuanto que la correspondencia entre la lista de los datos especificados en la CALL del programa que realiza la llamada y en USING del programa llamado es rígida, con una correspondencia precisa de la primera variable con la primera variable del otro programa, de la segunda con la segunda, etc.

Es fundamental que todos los datos definidos en PROCEDURE DIVISION USING... del programa llamado sean declarados en alguna sección de su DATA DIVISION.

La instrucción EXIT PROGRAM restituirá el control al programa que hace la llamada.

También para CALL es posible que un programa llamado llame, a su vez, a otro programa, transfiriéndole datos y valores.

CAPITULO X

COBOL: EL ACCESO A LOS FICHEROS

File Status

Se vio anteriormente que en la sección FILE CONTROL es posible definir una variable en la que encontrar el resultado de una operación de acceso a los datos de los ficheros. Esta variable "de estado" comprende dos caracteres, de los cuales el primero indica si la operación tuvo un resultado satisfactorio o no y el segundo depende del valor tomado.

Si el primer carácter contiene un 0, ello indica que la operación tuvo un resultado satisfactorio y, por consiguiente, también el segundo carácter tendrá un 0.

Si el primer carácter contiene el valor 1, el programa encontró una tentativa de acceso a registros externos al fichero, es decir, la condición de fin_del_fichero. Este estado se indica con AT END, por cuanto que puede determinarse con instrucciones especiales. También en este caso el segundo carácter contendrá 0.

Si el primer carácter contiene 2, lo que no se puede verificar con ficheros de organización secuencial, el programa señala la especificación de una clave de acceso no válida, condición que se expresa por INVALID KEY, que puede determinarse con instrucciones sucesivas. También en este caso el segundo carácter contendrá un valor 0.

Si el primer carácter contiene 3, el programa registró un error no recuperable, que puede depender de los datos, de las operaciones de entrada/salida, etc. El segundo carácter contendrá todavía un valor 0.

Si el primer carácter contiene 9, el programa registró un error que suele ser particular del sistema operativo del ordenador, es decir del ambiente que "rodea" al programa COBOL. En este caso, el segundo carácter contendrá un código que indica la naturaleza del error encontrado.

OPEN

Para tener acceso a un fichero, la primera operación a efectuar es la de "abrirlo", es decir, de hacerlo accesible al programa introduciéndolo en un canal de comunicación con el ordenador.

Al tener relación con ficheros de acceso secuencial, la sintaxis será:

```
OPEN INPUT  nombre_del_fichero_1 , ...   [WITH NO REWIND]
     OUTPUT nombre_del_fichero_2 , ...   [WITH NO REWIND]
     EXTEND nombre_del_fichero_3 , ...
```

(varios ficheros pueden abrirse de forma simultánea).

Si la operación de apertura tuvo un resultado satisfactorio, se tendrá la posibilidad de acceder a los datos contenidos en el fichero. En el curso del mismo programa no es posible reabrir un fichero todavía abierto, a no ser que se provoque un error; se deberá cerrar con anterioridad.

Si un fichero se abre en INPUT el programa se posicionará de forma automática en su primer registro, por cuanto que el acceso secuencial permite solamente la lectura de los registros uno tras otro, en orden de presencia en el fichero. La opción NO REWIND, que solamente es válida cuando el fichero tiene el soporte de la unidad de almacenamiento en cinta, no da lugar a este reposicionamiento automático, que tendrá que ejecutarse mediante otras instrucciones.

También con un fichero abierto en OUTPUT, el programa apuntará a su primer registro (a no ser que sea especificado NO REWIND) para iniciar las operaciones de escritura. Es importante destacar el hecho de que la apertura de un fichero en OUTPUT, si se utilizara con un fichero no existente, realiza su "creación", mientras que si el fichero existe ya, efectuará el borrado completo de su contenido anterior.

La opción EXTEND indica la intención de efectuar operaciones de escritura en el fichero a partir de su último registro, es decir, extendiendo su dimensión sin destruir el contenido anterior.

Para ficheros de acceso directo, relativos o de índice, la sintaxis será:

```
OPEN INPUT   nombre_del_fichero_1 , ...
     OUTPUT  nombre_del_fichero_2 , ...
     I-O     nombre_del_fichero_3 , ...
```

por cuanto que las opciones EXTEND y NO REWIND no tienen sentido para ficheros con tal organización, mientras que es importante, precisamente por la naturaleza de estos ficheros, tener la posibilidad de leer y escribir simultáneamente en el mismo registro.

Para estas clases de ficheros, la apertura no da lugar al posicionamiento en el primer registro, por cuanto que para cada operación de acceso a los datos se tendrá que especificar (mediante el valor de RELATIVE y RECORD KEY) qué registro abordar.

Para estos tipos de fichero, la indicación de INPUT en presencia de LABEL RECORD, determina la evaluación de los valores de las etiquetas, mientras que OUTPUT da lugar a la nueva escritura de las etiquetas "LABEL" según las especificaciones necesarias.

Además, si se "direcciona" un OPEN OUTPUT a un fichero no existente se producirá su creación, mientras que OPEN INPUT para un fichero inexistente dará lugar, con la primera tentativa a los datos, a una condición de error AT END.

Por último, cualquier operación de OPEN producirá la actualización del valor de la variable STATUS.

CLOSE

Cuando estén terminadas todas las operaciones de acceso a uno o más ficheros, es conveniente "apartarlos" del programa mediante la instrucción CLOSE.

Para los ficheros secuenciales su sintaxis será:

```
CLOSE nombre_del_fichero_1 [REEL WITH NO REWIND
                            UNIT
                            LOCK]
      nombre_del_fichero_2 ...
      ...
```

mientras que para los ficheros de acceso directo, relativos y con índice, tendremos:

```
CLOSE nombre_del_fichero_1 [LOCK]
      ...
```

en donde REEL y UNIT tienen una función exclusiva de documentación sobre las características de las unidades de cintas.

La opción LOCK tiene una importancia especial, por cuanto que impide que, en el curso del programa, se pueda volver a abrir el fichero correspondiente. Por supuesto, es posible efectuar una operación de CLOSE solamente en ficheros abiertos con anterioridad.

Una función especial de CLOSE es la de escribir en el fichero la señal de End_Of_File (final de fichero), reconocida como un carácter especial por los programas.

La instrucción STOP RUN ejecutada sin CLOSE obligará, por el contrario, al sistema operativo del ordenador a cerrar el fichero sin emitir la señal de EOF (final de fichero).

También la operación de CLOSE actualizará el valor de FILE STATUS.

Acceso a los registros

El único modo de acceder a un fichero "LOCKed" (bloqueado) por un programa es "lanzar", mediante otro programa, una instrucción tal como:

UNLOCK nombre_del_fichero [RECORD]

que hace disponible el último registro procesado de un fichero bloqueado.

Por supuesto, esta instrucción no permite el acceso al fichero por parte del mismo programa que lo ha bloqueado.

De cualquier modo, vamos a examinar las operaciones más comunes de acceso a los datos de los ficheros.

En lo que respecta a la escritura, se utiliza el comando WRITE que, en el caso de ficheros de organización secuencial, tiene la sintaxis:

WRITE nombre_del_registro [FROM nombre_1]
BEFORE/AFTER ADVANCING PAGE/nombre_2 LINES

El efecto de dicha instrucción será realizar una escritura en el fichero (a condición de que esté abierto OUTPUT o EXTEND) utilizando como destino la definicción del nombre_del_registro proporcionada en DATA DIVISION.

Si existe la opción FROM, el efecto será transferir el valor de nombre_1 a nombre_del_registro (como por efecto de una instrucción MOVE).

La opción ADVANCING, válida solamente si se selecciona la impresora, especifica la ejecución de una impresión del registro que deberá ir seguida o precedida por el avance de una serie de líneas o de una página completa. En lo que respecta a los límites del fichero, puesto que se fijaron en otra sección, no pueden modificarse por una simple operación de escritura y, por consiguiente, en el caso de superación por el registro de las dimensiones del fichero, se producirá una situación de error, verificable en FILE STATUS.

En caso de ficheros relativos o con índice, la sintaxis será:

WRITE nombre_del_registro [FROM nombre_1]
INVALID KEY instr_1

Para los ficheros relativos y con índice, si el acceso es secuencial, no será necesario indicar el valor de RECORD KEY, que será proporcionado directamente por el sistema operativo. Es diferente el caso de acceso aleatorio o dinámico o de apertura I-O (entrada/salida) por cuanto que el programa deberá establecer la clave de acceso al registro que se quisiere escribir. Por consiguiente, mientras que en el acceso secuencial los registros se procesarán uno tras otro en el mismo orden "lógico" en el que se dispusieron en el registro, con los accesos dinámico y aleatorio será posible que los registros se escriban en cualquier orden, sin la obligación de que sean consecutivos.

La indicación de INVALID KEY es necesaria cuando no exista un procedimiento USE (de tratamiento de errores) asociado al fichero. Se hace operativa cuando existe un error en la especificación de una clave de acceso al fichero (clave ya asignada, registro existente, etc.) o si se intentan superar los límites del fichero. En este caso el control pasará a instr_1. Por supuesto, el valor de FILE STATUS registrará esta condición.

Otra instrucción de escritura, que se utilizará cuando se quiera sustituir el valor de un registro ya existente, es:

REWRITE nombre_del_registro [FROM nombre_1]

en caso de acceso secuencial y con la adición de:

INVALID KEY instr_1

para acceso relativo y con índice.

Por supuesto, deberá declararse la modalidad I-O para permitir dicha escritura, la cual ha de ir precedida por una operación de READ (lectura) en el mismo fichero.

En lo que respecta a la lectura de un registro de un fichero,

la instrucción utilizada es READ que, en caso de acceso secuencial, tiene la sintaxis:

```
READ nombre_del_fichero RECORD [INTO nombre_1]
        [AT END instr._1]
```

Realiza la lectura en correspondencia con la posición del "puntero", que es la señal que indica el registro en el que se ejecutará la operación sucesiva. Esto implica que, si se acaba de efectuar una operación de OPEN secuencial, el fichero fue objeto de posicionamiento en el primer registro y, por consiguiente, se leerá el primer dato del fichero, mientras que si se realizaron ya otras operaciones de lectura "READ", al ser automático el desplazamiento del puntero, el registro leído será el siguiente al último procesado.

La opción INTO produce un efecto similar a la instrucción MOVE y tiene efecto solamente si READ fue objeto de referencia a registros de dimensiones fijas y si la operación tuvo un resultado satisfactorio.

La opción AT END, a especificar cuando no se asoció una subrutina de USE al fichero, indica la instrucción a la que se deberá transferir el control del programa cuando se intente leer más allá del último registro del fichero, es decir, cuando se haya detectado el final de fichero (EOF). Será, pues, necesario, para tener acceso de nuevo al fichero, una secuencia de CLOSE y OPEN. Con ficheros relativos y con índice, se utilizará:

```
READ nombre_del_fichero [NEXT RECORD WITH NO LOCK]
     [INTO nombre_1]
     AT END instr._1
```

o bien

```
READ nombre_del_fichero RECORD [WITH NO LOCK]
     [INTO nombre_1]
     KEY IS nombre_dato
     INVALID KEY instr._2
```

La primera forma se usa cuando se quieren leer de manera secuencial los ficheros de acceso dinámico y la segunda cuando se quieran leer de modo aleatorio.

La opción KEY IS sólo es necesaria cuando el fichero es del tipo con índice, por cuanto que se precisa especificar, para cada operación, la clave dc acceso a los registros.

Por consiguiente, AT END e INVALID KEY representan la indicación de la subrutina de control de la situación particular que se va a crear cuando se está en el final de fichero (para acceso secuencial) o si se indica una clave de acceso errónea (de tipo aleatorio).

Para que la primera forma de READ tenga sentido es necesario que el puntero del registro a leer esté posicionado con la instrucción START. Ello será necesario solamente para la primera lectura, por cuanto que el puntero actualiza su posición después de cada acceso.

En caso de fichero con índice, cuando un registro se haya descrito con varias estructuras, puesto que comparten la misma zona, sus datos estarán disponibles simultáneamente en su totalidad.

Si un fichero es relativo y se especificó la opción RELATIVE KEY, un acceso a un registro, realizado según la primera forma, introducirá en RELATIVE KEY el número del registro leído. Por el contrario, para ser operativa la segunda forma se requiere la especificación del valor de la clave de acceso al registro que, si no existe, dará lugar a la activación de la instrucción INVALID KEY.

A falta de especificación de la clave (KEY) se utilizará la que corresponde al primer registro del fichero.

Si un fichero es del tipo con índice y existen claves de acceso duplicadas, los registros correspondientes se procesarán en el mismo orden en que fueron escritos o reescritos.

Con este tipo de fichero el acceso según la primera forma introducirá en RECORD KEY el valor correspondiente al registro procesado, mientras que, por el contrario, con la segunda forma es obligatorio especificar la clave del registro al que se quiere tener acceso. A falta de esta especificación se partirá del registro que corresponde al primer valor de la lista de las claves.

Por consiguiente, una función especial es la que desempeña la instrucción START, que permite el "apuntamiento" al registro particular especificado por el valor de KEY. Su sintaxis, para ficheros relativos y con índice, es:

```
START nombre_fichero   KEY IS EQUAL   TO   nombre_dato
                           IS =
                           IS GREATER THAN
                           IS         >THAN
                           IS NOTLESS THAN
                           IS NOT <
        INVALID KEY   instr._1
```

en donde el valor de la clave RELATIVE KEY, válida para el acceso al registro del fichero, se especifica por la comparación condicional indicada.

A falta de especificaciones, se sobrentiende el valor EQUAL TO (IGUAL A) y se refiere al primer valor de la lista de las claves de acceso.

La comparación se realiza siguiendo las reglas de las comparaciones alfabéticas, es decir, según el orden de la COLLATING SEQUENCE (ORDEN LEXICOGRAFICO DE CLASIFICACION DE LOS CARACTERES) y, en caso de comparaciones entre elementos de diferente longitud, se limita al examen del número de caracteres menor entre los dos.

La última operación de acceso a un fichero es la de borrado de un registro de un fichero abierto en I-O. Su sintaxis es:

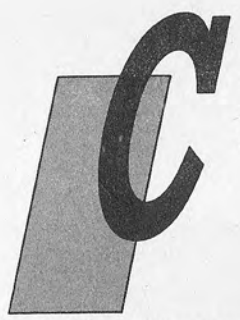
```
DELETE nombre_del_fichero  RECORD
       INVALID KEY instr_1
```

La instrucción, después de READ, no afecta a la zona de memoria asociada con el registro, pero imposibilita su acceso lógico en operaciones sucesivas.

Para ficheros de acceso secuencial el registro borrado es el último leído, mientras que para ficheros de acceso aleatorio será el especificado por el valor de la clave (KEY).

CAPITULO XI

COBOL: PROGRAMAS DE EJEMPLO



Consideremos como programas de ejemplo uno que realice la carga de datos de almacén y otro que imprima los registros relativos.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.ES-2.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE COMPUTER.
OBJECT COMPUTER.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT ALMACEN
                 ASSIGN TO RANDOM,"TARJETAS"
                 ORGANIZATION INDEXED
                 ACCESS SEQUENTIAL
                 RECORD KEY CODIGO.
       SELECT PAGINA
                 ASSIGN TO PRINT.

DATA DIVISION.

FILE SECTION.
FD ALMACEN
                 RECORD 32 CHARACTERS
                 DATA RECORD ARTICULO.
01 ARTICULO.
   02 CODIGO          PIC X(6).
   02 DENOMINACION    PIC X(20).
   02 UNIDAD-MEDIDA   PIC X(6).
```

```
FD PAGINA
                  RECORD 80 CHARACTERS
                  DATA RECORD LINEA.
01 LINEA.
   02 NUM-ORDEN        PIC 9(6).
   02 FILLER           PIC X(4).
   02 PCODIGO          PIC X(6).
   02 FILLER           PIC X(4).
   02 PDENOM           PIC X(20).
   02 FILLER           PIC X(4).
   02 PUNIDAD          PIC X(6).
   02 FILLER           PIC X(30).
01 ROTULO REDEFINES LINEA.
   02 CABECERA         PIC X(50).
   02 NUM-PAG          PIC X(4).
   02 FILLER           PIC X(26)

WORKING-STORAGE SECTION.
   77 PROGR            PIC 9(6)  VALUE 0.
   77 CONT-LINEAS      PIC 9(2)  VALUE 1.
   77 NPAG             PIC 9(6)  VALUE 0.
   77 MAX-PAG          PIC 9(2)  VALUE 60.
   77 CONFIRMACION     PIC X.

PROCEDURE DIVISION.

INICIO.
     OPEN INPUT ALMACEN
          OUTPUT PAGINA.
     DISPLAY SPACE.
     MOVE SPACE TO LINEA.
SOLICITUD-CONFIRMACION.
     DISPLAY "QUIERE REALIZAR LA IMPRESION?".
     ACCEPT CONFIRMACION.
     IF CONFIRMACION EQUAL TO "N" OR CONFIRMACION
                     EQUAL TO SPACE GO TO FIN.
     MOVE "              ALMACEN              Pagina"
                     TO CABECERA.
PREPARACION-IMPRESION.
     PERFORM TITULO THRU FIN-TITULO.
LECTURA-FICHERO.
     READ ARTICULO AT END GO TO FIN.
     ADD 1 TO PROGR,CONT-LINEAS.
     IF CONT-LINEAS GREATER THAN MAX-PAG
                     PERFORM TITULO THRU FIN-TITULO.
     MOVE PROGR TO NUM-ORDEN.
     MOVE CODIGO TO PCODIGO.
     MOVE DENOMINACION TO PDENOM.
     MOVE UNIDAD-MEDIDA TO PUNIDAD.
     WRITE LINEA AFTER ADVANCING 1 LINE.
     GO TO LECTURA-FICHERO.
TITULO.
     ADD 1 TO NPAG.
```



```
     MOVE NPAG TO NUM-PAG.
     WRITE ROTULO AFTER ADVANCING PAGE.
     MOVE 0 TO CONT-LINEAS.
FIN-TITULO.
     EXIT.
FIN.
     CLOSE ALMACEN PAGINA.
     DISPLAY SPACE.
     DISPLAY "FIN IMPRESION".
     STOP RUN.


IDENTIFICATION DIVISION.
PROGRAM-ID.ES-1.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE COMPUTER.
OBJECT COMPUTER.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT ALMACEN
                     ASSIGN TO RANDOM,"TARJETAS"
                     ORGANIZATION INDEXED
                     ACCESS DYNAMIC
                     RECORD KEY CODIGO.

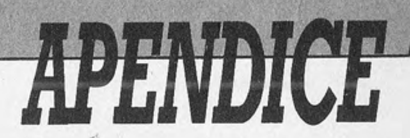
DATA DIVISION.
FILE SECTION.
FD ALMACEN
                     RECORD 32 CHARACTERS
                     DATA RECORD ARTICULO.
01 ARTICULO.
   02 CODIGO           PIC X(6).
   02 DENOMINACION     PIC X(20).
   02 UNIDAD-MEDIDA    PIC X(6).

WORKING-STORAGE SECTION.
01 VISUALIZACION.
   02 INPUT-CODIGO     PIC X(40)
      VALUE "CODIGO                       ........".
   02 FILLER           PIC X(120).
   02 INPUT-DENOM      PIC X(40)
      VALUE "DENOMINACION                 ........".
   02 FILLER           PIC X(120).
   02 INPUT-MEDIDA     PIC X(40)
      VALUE "UNIDAD DE MEDIDA             ........".
01 PANTALLA-INPUT REDEFINES VISUALIZACION.
   02 FILLER           PIC X(34).
   02 ICODIGO          PIC X(6).
   02 FILLER           PIC X(140).
```

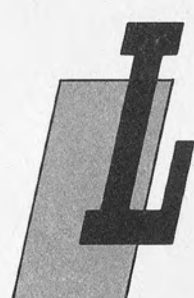
```
    02 IDENOM              PIC X(20).
    02 FILLER              PIC X(154).
    02 IMEDIDA             PIC X(6).

PROCEDURE DIVISION.

INICIO.
     OPEN I-O ALMAÇEN.
     DISPLAY SPACE.
     DISPLAY VISUALIZACION.
CARGA.
     DISPLAY PANTALLA-INPUT.
     ACCEPT PANTALLA-INPUT.
     IF CODIGO EQUAL TO "000000" GO TO FIN.
     IF IDENOM EQUAL TO SPACE GO TO CARGA.
     IF IDENOM NOT EQUAL TO SPACE AND IMEDIDA EQUAL
                 TO SPACE GO TO CARGA.
     MOVE ICODIGO TO CODIGO.
     MOVE IDENOM TO DENOMINACION.
     MOVE IMEDIDA TO UNIDAD-MEDIDA.
     GO TO CARGA.
FIN.
     CLOSE ALMACEN.
     DISPLAY SPACE.
     DISPLAY "FIN CARGA".
     STOP RUN.
```

Funciones intrínsecas del Fortran



os parámetros de las funciones están indicados por las abreviaturas siguientes:

I	entero
R	real
D	doble precisión
C	complejo
S	carácter

Nombre de la función	Tipo	Definición
ABS(I) o IABS(I)	Entero	Valor absoluto:I:
ABS(R)	Real	Valor absoluto:R:
ABS(D) o DABS(D)	D.Prec.	Valor absoluto:D:
ABSC(C)	Real	Raíz cuadrada (REAL(C) +AIMAG(C))
ACOS(R)	Real	Arco seno (R)
ACOS(D) o DACOS(D)	D.Prec.	Arco seno (D)
AINT(R)	Real	Truncación REAL(INT(R))
AINT(D) o DINT(D)	D.Prec.	Truncación REAL(INT(D))
ANINT(R)	Real	Entero próximo REAL(INT(R+/-0,5))
ANINT(D) o DNINT(D)	D.Prec.	Entero próximo REAL(INT(D+/-0,5))
ASIN(R)	Real	Arco seno (R)
ASIN(D) o DASIN(D)	D.Prec.	Arco seno (D)
ATAN(R)	Real	Arco tangente (R)
ATAN(D) o DATAN(R)	D.Prec.	Arco tangente (D)
ATAN2(R1,R2)	Real	Arco tangente (R1/R2)
ATAN2(D1,D2) o DATAN2(D1,D2)	D.Prec.	Arco tangente (D1/D2)
CMPLX(I)	Compl.	Complejo (REAL(I),0)
CMPLX(R)	Compl.	Complejo (REAL(R),0)

Nombre de la función	Tipo	Definición
CMPLX(D)	Compl.	Complejo (REAL(D),0)
CMPLX(I1,I2)	Compl.	Complejo (REAL(I1),REAL(I2)
CMPLX(R1,R2)	Compl.	Complejo (REAL(R1),REAL(R2)
CMPLX(D1,D2)	Compl.	Complejo (REAL(D1),REAL(D2)
CMPLX(C)	Compl.	Complejo C
COS(R)	Real	Coseno (R)
COS(D) o DCOS(D)	D.Prec.	Coseno (D)
COS(C) o CCOS(C)	Compl.	Coseno (C)
COSH(R)	Real	Cos. hiperb. (R)
COSH(D) o DCOSH(D)	D.Prec.	Cos. hiperb. (D)
DBLE(I)	D.Prec.	D.Prec. de REAL(I)
DBLE(R)	D.Prec.	D.Prec. de REAL(R)
DBLE(C)	D.Prec.	D.Prec. de REAL(C)
DBLE(D)	D.Prec.	D.Prec. D
DIM(I1,I2) o IDIM(I1,I2)	Entero	Dif.positiva MAX((I1-I2),0)
DIM(R1,R2)	Real	Dif.positiva MAX((R1-R2),0)
DIM(D1,D2) o DDIM(D1,D2)	D.Prec.	Dif.positiva MAX((D1-D2),0)
EXP(R)	Real	e^R
EXP(D)	D.Prec.	e^D
EXP(C)	Compl.	e^C
FLOAT(I)	Real	REAL(I)
IFIX(R)	Entero	INT(R)
INT(R)	Entero	Entero de REAL(I)
INT(D)	Entero	Entero de REAL(D)
INT(C)	Entero	Entero de REAL(C)
INT(I)	Entero	Entero I
LOG(R)	Real	Log.natural (R)
LOG(D) o DLOG(D)	D.Prec.	Log.natural (D)
LOG(C) o CLOG(C)	Compl.	Log.natural (C)
LOG10(R)	Real	Log.decimal (R)
LOG10(D) o DLOG10(D)	D.Prec.	Log.decimal (D)
MAX(I1,I2) o MAXO(I1,I2)	Entero	Máximo entre I1 e I2
MAX(R1,R2)	Real	Máximo entre R1 y R2
MAX(D1,D2) o DMAX(D1,D2)	D.Prec.	Máximo entre D1 y D2
MIN(I1,I2) o MINO(I1,I2)	Entero	Mínimo entre I1 e I2
MIN(R1,R2) o	Real	Mínimo entre R1 y R2
MIN(D1,D2) o DMIN(D1,D2)	D.Prec.	Mínimo entre D1 y D2
MOD(I1,I2)	Entero	Resto: I1-INT(I1/I2)*I2
MOD(R1,R2) o AMOD(R1,R2)	Real	Resto: R1-INT(R1/R2)*R2
MOD(D1,D2) o DMOD(D1,D2)	D.Prec.	Resto: D1-INT(D1/D2)*D2
NINT(R)	Entero	Entero próximo INT(ANINT(R))
NINT(D) o IDNINT(D)	Entero	Entero próximo INT(ANINT(D))
REAL(I)	Real	Real de I
REAL(D)	Real	Real de D
REAL(R)	Real	Real de R
REAL(C)	Real	Real de C
SIGN(I1,I2) o ISIGN(I1,I2)	Entero	I1 si I2>0, -I1 si I2<0
SIGN(R1,R2)	Real	R1 si R2>0, -R1 si R2<0
SIGN(D1,D2) o DSIGN(D1,D2)	D.Prec.	D1 si D2>0, -D1 si D2<0
SIN(R)	Real	Seno (R)
SIN(D) o DSIN(D)	D.Prec.	Seno (D)
SIN(C) o CSIN(C)	Compl.	Seno (C)
SINH(R)	Real	Seno hiperb. (R)
SINH(D) o DSINH(D)	D.Prec.	Seno hiperb. (D)
SINH(C) o CSINH(C)	Compl.	Seno hiperb. (C)

Nombre de la función	Tipo	Definición
SNGL(D)	Real	REAL(D)
SQRT(R)	Real	Raíz cuadrada (R)
SQRT(D) o DSQRT(D)	D.Prec.	Raíz cuadrada (D)
SQRT(C) o CSQRT(C)	Compl.	Raíz cuadrada (C)
TAN(R)	Real	Tangente (R)
TAN(D) o DTANH(D)	D.Prec.	Tangente (D)
TANH(R)	Real	Tangente hiperb.(R)
TANH(D) o DTANH(D)	D.Prec.	Tangente hiperb.(D)
AIMAG(C)	Real	Parte imaginaria de C
AMAXO(I1,I2,...,In)	Real	REAL(MAX(I1,I2,...,In))
AMINO(I1,I2,...,In)	Real	REAL(MIN(I1,I2,...,In))
CHAR(I)	Caract.	Carácter correspondiente a I
CONJG(C)	Compl.	Conjugado: REAL(C),-AIMAG(C)
DPROD(R1,R2)	D.Prec.	R1*R2 en Doble Precisión
ICHAR(S)	Entero	Nº de orden de S en la tabla
INDEX(S1,S2)	Entero	Posición subcadena S2 en S1
LEN(S)	Entero	Nº de caracteres de S
LGE(S1,S2)	Lógico	Verdadero si S1 > 0 = S2
LGT(S1,S2)	Lógico	Verdadero si S1 > S2
LLE(S1,S2)	Lógico	Verdadero si S1 < 0 = S2
LLT(S1,S2)	Lógico	Verdadero si S1 < S2
MAX1(R1,R2,...,Rn)	Entero	INT(MAX(R1,R2,...,Rn))
MIN1(R1,R2,...,Rn)	Entero	INT(MIN(R1,R2,...,Rn))

NOTAS



Frente a la generalidad de otros lenguajes, como el archiconocido BASIC, el FORTRAN y el COBOL presentan una gran especialización. Esto supone ventajas e inconvenientes, indudablemente. Por un lado, dificulta su conocimiento al gran público y restringe su campo de acción, pero, por otro, los hace muy recomendables en ciertas aplicaciones.

Si tiene interés en saber algo más acerca de los lenguajes mayoritariamente usados en las aplicaciones de cálculo científico (FORTRAN) y comercial (COBOL) lea el contenido de este volumen de la B.B.I.

395 pts.
(incluido IVA)